apricot F1

Technical Reference

Copyright

Portions of this manual contain material reprinted by permission of:

SONY Corporation, Copyright 1983

Trademarks

MS-DOS and Microsoft are registered trademarks of the Microsoft Corporation.

CP/M is a registered trademark of Digital Research.

It is possible that this manual may contain references to or describe Apricot products which are not available in your country. Such references or information should not be taken as an understanding that these products will become available.

Information contained in this document is subject to change without notice and does not represent a commitment on the part of Apricot.

All rights reserved, no use or disclosure without written consent.

Copyright © Apricot Computers plc

Published in the U.K. by

Apricot UK Ltd
Shenstone House
Dudley Road
Halesowen
West Midlands B63 3NT

Published in the USA by

Apricot Inc.
47173 Benicia Street
Fremont
California 94538

Preface

The Technical Reference Manual for the ACT Apricot F1 microcomputer is intended for programmers and engineers involved in hardware and software design for the F1.

The Manual is divided into three sections and a number of appendices as detailed below.

1. Overview

This section provides an overall description of the F1 and is sub-divided into three chapters:

System Overview

This chapter presents an overall picture of the F1 concentrating on mainly the hardware, but also including information on some of the software aspects which are integral to the machines' operation.

Software

This chapter provides a brief description of the operating system and it's interface to the associated BIOS. An introduction to the software modules of the BIOS is also provided.

Options

This chapter forms the introduction to all the other ACT hardware options available for extending the capabilities of any machine within the range.

2. Hardware Detail

This section contains detailed descriptions of all the hardware aspects of the microcomputer and is divided into a number of chapters, as detailed in the following *Contents* section.

The Systems Unit and the Keyboard of the F1 are discussed in detail with major programmable elements (e.g. serial port, display circuitry, etc.) also having separate descriptions.

3. Software Detail

This section contains a detailed description of all software aspects of the BIOS and is also divided into a number of chapters. The first provides a detailed description of the BIOS as a whole. Subsequent chapters detail the features and facilities of the individual hardware device drivers.

Appendices

A number of appendices are included in this manual which provide general hardware reference information and also associated software information of specific use to systems/application programmers.

Addenda

This section details installable device drivers supplied as system software and will also be expanded to reflect updates and changes as the product continues to be developed.

Associated Publications

MICROSOFT MS-DOS Programmer's Reference Guide

This is an absolute necessity for anybody who wants to develop software for the Apricot. It provides details of:

1. MS-DOS 2.00 system calls and interrupts.
2. How to produce and install new device drivers.
3. MS-DOS memory maps, disk layout, DOS initialisation and many more technical details.

The UK source of this product is:

Order Processing Dept
Microsoft Ltd, Piper House
Hatch Lane, Windsor, BERKS.

DIGITAL RESEARCH GSX-86 Graphics Extension Programmer's Guide

This is a necessity for any programmer who wishes to produce graphics based packages interfacing to Apricot's implementation of the Digital Research GSX module. This product can be obtained through your local ACT dealer.

The 8086 Family User's Manual

This provides descriptive material on the Intel 8086 processor and includes all the necessary details on the instruction sets for asssembly language programmers. The manual can be obtained from an Intel distributor.

Contents

1. Overview

1.1 System Overview

2 **Introduction**

3 **Details**

3 Packaging and styling
6 Processing Capability
8 Memory
9 Disk Drives
10 Display Features
13 Keyboard
17 Printer Support
18 Communications
20 Expansion
21 Specification

1.2 Software Overview

2 **Introduction**

4 **Details**

4 Applications Interface
8 Operating Systems Interface
9 MS-DOS
10 ROM BIOS

1.3 Options

2 **Introduction**

3 **Details**

3 Display Options
5 RAM Expansion Boards
6 Modem Board
8 LAN Board
9 Mouse
10 Apricot MSD
11 Expansion Unit

2. Hardware Detail

2.1 Systems Unit

2 **Introduction**

3 **Details**
3 Mechanics
4 Connectors
6 Front Panel
6 System Board
8 Infra-Red Detector Board
9 Disk Drive Unit
9 Expansion
10 Power Supply
12 Physical dimensions

2.2 System Detail

2 **Introduction**

3 **Details**
3 General
6 Processor
8 Memory
10 Interrupt Control
11 Display Control
15 Floppy Disk Control
16 Expansion
17 Keyboard/Mouse data
18 System Reset
19 RS232 Communications
20 Parallel Printer Port
21 System Timer
21 Sound Generation
22 Port Addresses

2.3 Interrupt Control

2 **Introduction**

5 **Details**
5 General
6 Maskable Interrupt vectors
7 Non-Maskable Interrupt (NMI)
7 Programming the controllers
9 Interrupt Control Sequence
10 Maskable Interrupts

13 **Programming**

2.4 Display Control

2 **Introduction**

3 **Details**
3 General
4 Display modes and Features
6 Drive Signals
7 Circuitry
9 Display RAM
12 Pointer RAM
15 Palette RAM
24 Mode Selection
25 Refresh control and Timing
29 Display Connectors

2.5 Expansion

2 **Introduction**

3 **Details**
3 General
4 Expansion Slot
7 Expansion Connector
8 Electrical Specification
9 Pin Detail
12 Address Allocation
13 Using Interrupts
16 Expansion Board Layout

2.6 Floppy Disk Interface

2 **Introduction**

3 **Details**
3 General
5 Disk Write
6 Disk Read
7 Disk formatting
8 Read/write head positioning

10 **FDC detail**
10 General
11 Processor Interface
11 Data Requests
12 Interrupt Requests
14 Disk Drive Control
15 Command Register
16 Status Register
16 Track Register
16 Data Register

17 **Programming Considerations**
17 General
18 Disk Drive Selection
18 Motor Control
19 Head Loading
19 Head Positioning
23 Data Transfers
28 Formatting Commands
33 Force Interrupt Command

35 **Interface Connection Detail**
35 System Connections
37 Disk Drive Unit Connections

39 **Track Format**

2.7 Serial Interface

2 **Introduction**

4 **Details**
4 General
5 SIO Overview
7 SIO Architecture
10 Processor Interface
12 Write Register Definition
13 Write Register Summary
14 Write Register 0
17 Write Register 1
19 Write Register 2
20 Write Register 3
21 Write Register 4
23 Write Register 5
25 Write Register 6
25 Write Register 7
25 Read Register Definition
26 Read Register 0
28 Read Register 1
29 Read Register 2

30 **SIO Interrupt Sequence**

32 **Keyboard/Mouse Data**

34 **Sound Generation**

35 **Channel A Programming Details**
35 Copy Registers
36 Initialisation
37 Generating Sound

38 **RS232C Communications**
38 General
39 RS232C Connector Detail

40 **Channel B Programming Details**
40 Copy Registers
41 Setting the Base Vector
41 Asynchronous Communications

45 **SIO Pin Detail**
45 System Connections
47 Channel A Connections
49 Channel B Connections

2.8 Printer Interface

2 **Introduction**

3 **Details**
3 General
4 Data Transfers
5 Connector Detail
6 Address Allocation

7 **Programming Considerations**
7 Data Port
7 Printer Status
7 Data Strobe

2.9 Timer

2 **Introduction**

4 **Details**
4 General
4 Channel modes
5 Clock rates
5 Interrupts
7 Channel usage
8 Address allocation

9 **Programming Considerations**
9 Initialisation
10 Setting the base interrupt vector
10 Channel 0: Expansion Interrupts
11 Channel 1: RS232C baud rate
14 Channel 2: Sound Frequency
16 Channel 3: System Clock
16 Return from Interrupt Sequence

2.10 Sound Generation

2 **Introduction**

4 **Details**
4 General
5 Generating Sound
6 Address allocation

7 **Programming Considerations**
7 General
8 Initialisation
8 Simple tones
10 Complex sounds

2.11 Disk Drive

2 **Introduction**

4 **Details**
4 General
4 Interface Details
6 Interface Connections (Outputs)
7 Interface Connections (Inputs)
9 Disk Drive Mechanism
9 Read/Write Heads
9 Head Positioning Mechanism
9 Head Load Mechanism
10 Sensors and Detectors
10 Drive Switch Settings
12 Drive Specification

13 **Disks**
13 General
13 Disk Precautions
14 Disk Insertion/Removal
14 Write Protecting
14 Disk Format

2.12 Keyboard

2 **Introduction**

3 **Details**
3 Mechanics
4 Circuitry
4 Keyboard Scanning
5 Data Transmission Format
6 Keycode Data Encoding
11 Special Keys

Section 3: Software Detail

3.1 Guide to the BIOS

2 **Introduction**

3 **Bootstrap**

5 **Initialised drivers**

7 **Initialised BIOS**

8 **Built-in functions**

9 **Memory Map**
11 Software interrupts
16 Hardware interrupts
17 Pointers
20 ASCII and Bit Screen Images
20 RAM BIOS for MS-DOS

21 **Disk Label Sector and Configuration Table**

3.2 Control device

2 **Overview**

3 **General application**
3 Introduction
4 Low level Control device access
5 High level Control device access
7 Errors

8 **Specific application**
9 Device Numbers
10 Screen
12 Keyboard
14 Serial I/O
18 Parallel I/O
20 Mouse
21 Clock
22 Sound
24 Floppy disk
26 Winchester

3.3 Screen driver

2 **Overview**

3 **Application interest**
3 Screen images
4 Using ESCape sequences
6 Screen environment
7 Apricot compatible mode
9 Colour
13 ANSI ESCape sequences
15 Windows and Cursor addressing
16 Fonts
17 Ascii control codes
17 ESCape Sequence Table

33 **Systems interest**
33 Screen Bit Image
34 Character attributes
35 40 Column mode
35 Scrolling
36 Configuration table

3.4 Keyboard driver

2 **Overview**

6 **Application interest**
6 Changing the keyboard table
8 Implementing STRING keys
10 Changing the keyboard driver operation
12 Special Keys
13 Default STRINGS
14 Prefixes
14 User Interrupt (F9 hex)

15 **Systems interest**
15 Initialisation
16 Steering
17 Down-code handler
19 Queues
20 Configurator
21 Apricot compatibility
22 Configuration table

3.5 Serial I/O Driver

2 **Overview**

3 **Applications interest**
3 Configuring the Serial Driver
6 Generic differences
6 User Interrupts

8 **Systems interest**
8 Configuration table

3.6 Parallel I/O Driver

2 **Overview**
2 **Applications interest**
2 **Systems interest**

3.7 Clock Driver

2 **Overview**
2 **Applications interest**
3 **Systems interest**

3.8 Sound Driver

1 **Overview**
1 **Applications interest**
1 **Systems interest**

3.9 Disk Driver

2 **Overview**

3 **Applications interest**
3 Non-MSDOS systems
4 Drive types
4 Label Sector
6 MS-DOS format
7 Disk formats
8 Disk Swapping

9 **Systems interest**
9 Configuration data

3.10 GSX – GIOS Details

2 **Overview**

3 **Applications interest**
3 Display features
5 Calling GSX
6 Additions to GSX 1.3

18 **Systems interest**
18 System files

Appendices

A — Diagnostic Error Codes

B — Default Keyboard Table

C — Ascii codes

D — Circuit Diagrams

E — ESCape sequence reference table

2 Specials
3 Character attributes
4 Screen attributes
5 Colour
6 Cursor positioning
7 WP primitives
10 Driver environment
13 Keyboard interaction
15 Generic obsoletes

F — Language interfaces

2 **Overview**

3 **Interpretive Basic**
5 The Data Segment Register
6 Non-BIOS routine calls

7 **Compiled Basic**
7 The Data Segment Register
8 Non-BIOS routine calls

9 **'C' Programming Language**
9 The Data Segment Register
10 Non-BIOS routine calls

Index

Section 1
Overview

System Overview 1.1

Contents

Introduction

Details

Packaging and styling
Processing Capability
Memory
Disk Drives
Display Features
Keyboard
Printer Support
Communications
Expansion
Specification

Illustrations

1. Apricot F1.

Introduction

The F1 is a full function business microcomputer, complete with a minimum of 256 Kbyte of RAM and a single double-sided MicroFloppy disk drive. It contains a sophisticated ROM based BIOS and the capability to drive either a monochrome or a colour monitor in a variety of resolutions and modes.

The standard features found on the F1 micro are:

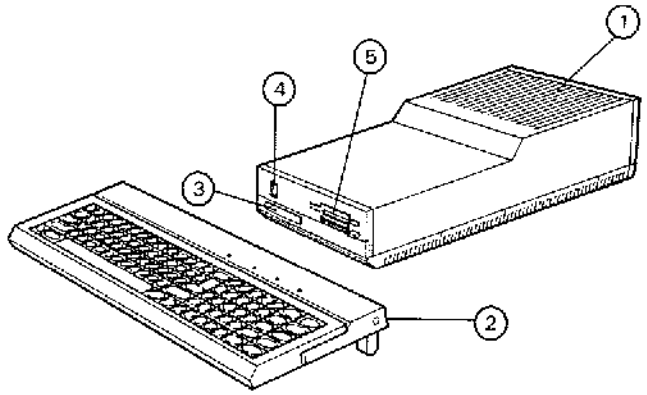
1. Parallel printer interface.
2. Asynchronous and synchronous RS232 communications capabilities.
3. Expandibility (via an ACT compatible expansion Slot and/or a separate Expansion Unit).
4. Low profile professionally styled Keyboard with QWERTY typewriter layout, calculator keypad and function keys.
5. Flexible display driver circuitry which allows the programmer to drive a colour monitor (optional) in a variety of resolutions and display modes. Alternatively, the same circuitry can be configured to drive a monochrome monitor or a standard domestic TV (requires the optional TV modulator).

Instead of being linked to the Systems Unit by a cable, the Keyboard uses an infra-red link for transmission of keycodes and other data (time and date information, hardware reset, etc). Multiple machine environments where infra-red interference could occur from other users are catered for by the use of a "light-pipe". This acts as a transmission line which directs the infra-red from the Keyboard to its parent Systems Unit.

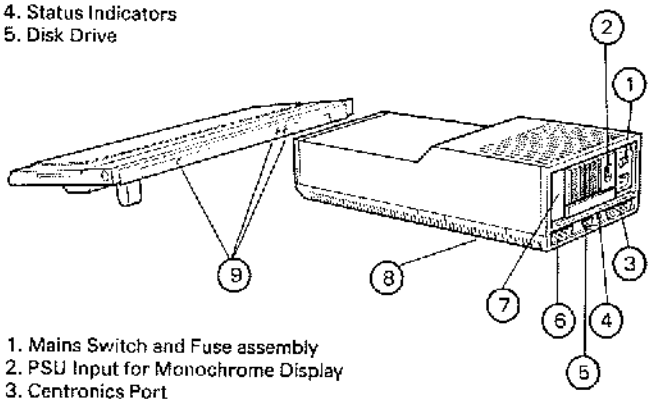
Details

Packaging and Styling

The F1 is composed of two main sections as previously mentioned; the Systems Unit and the Keyboard (see Figure 1). The machines are made into a full computer system by the addition of an appropriate display device. (A variety of different monitors are currently available as optional items from ACT. These include 9 inch and 12 inch monochrome monitors and a 10 inch colour monitor).



1. Systems Unit
2. Keyboard
3. IR Receivers
4. Status Indicators
5. Disk Drive



1. Mains Switch and Fuse assembly
2. PSU Input for Monochrome Display
3. Centronics Port
4. Composite Video Jack Socket
5. Colour/Monochrome Display Connector
6. RS232 Port
7. Expansion Plate
8. Expansion Cover
9. IR Transmitters

Figure 1. Apricot F1

The Systems Unit houses the majority of the control electronics, including the processing system and interfaces for:

1. A monochrome or colour monitor.
2. A standard domestic TV via an optional modulator.
3. Infra-red keyboard/mouse data.
4. Printers/plotters and similar devices.
5. External communications equipment.
6. Sound Generation.

It also contains the system RAM, a MicroFloppy disk drive, a power supply and a loudspeaker.

The Keyboard consists of a standard QWERTY typewriter section plus calculator keypad (with an identical key top layout to the Apricot pc/xi range of products), a block of 10 "fixed/programmable" function keys, and four recessed keys which perform specific fixed functions. It also includes:

1. Four AA batteries which form the power source for the keyboard electronics.
2. A processing system which provides the interface between the keys and the infra-red transmission circuitry.
3. A real time clock/calendar.

Freedom of user desk space was one of the criteria for using an infra-red cordless link between the Keyboard and the Systems Unit. This provides the user with a much greater flexibilty in positioning the Keyboard relative to the Systems Unit as compared with the standard cable link approach.

Four LED indicators are located on the front of the Systems Unit on the left-hand side of the disk drive slot. These provide the user with indications of various states within the machine, as specified by the associated indicator legend.

The indicators are normally illuminated to show the following states:

No Scroll	STOP key is active.
Caps	CAPS LOCK key is active.
Disk	The disk within the disk drive is being written to or read from.
Power	The machine is switched on.

Connections for a printer, a display monitor (monochrome or colour), and external communications equipment are all located on the back of the Systems Unit. Connections for the optional modulator for driving a standard domestic TV are located internally within the Unit. A connector on the right-hand side of the Unit is provided for linking in an optional Expansion Unit.

This is not the only method of expansion available within the machine. A single Expansion Slot is also located within the Systems Unit. This is electrically and physically compatible with all the current ACT Expansion Boards.

The mains switch is also located on the rear of the unit. This is part of a combined mains switch/fuse holder module. The fuse is located behind a hinged flap on the switch module housing.

A mains changeover link option is located internally within the Systems Unit to cater for the two standard mains operating voltages used throughout the world. This allows the F1 to be configured at the factory for either 110V or 240V operation.

To meet the desirable goal of quietness of operation, the F1 does not employ a fan for cooling (the major noise source on most other micros). Instead, cooling is provided by the natural airflow created in and around the machine by the positioning of vents in the case relative to the internal components which generate most of the heat.

Processing Capability

The F1 employs the Intel 8086 as the main central processing element. The features of the Intel 8086 are well known, being a true 16-bit processor, supported by the two major microcomputer operating systems companies (Microsoft with MS-DOS, Digital Research with CPM-86 and Concurrent DOS), and possessing:

1. 16-bit wide internal register architecture.
2. 16-bit wide external data bus.
3. Segmented addressing structure to support modular programming.
4. The capability of addressing up to 1 Mbyte of memory space and up to 64 Kbyte of system I/O.

The processing system operated on the board is a real time interrupt driven system, based upon the interrupt structure of the 8086 and the interrupt features provided by two Zilog chips; a Z80 SIO and a Z80 CTC. These two devices are daisy-chained and together supply a single interrupt line to the CPU. Hardware functions and processes which need interrupt facilities are in turn "connected" to these two devices (apart from the disk controller which uses the 8086 input NMI instead).

Each Zilog device posseses internal circuitry which perform the duties of an interrupt manager/arbitrator, making decisions to determine which hardware-driven process requires servicing by the 8086. The decisions are made on an fixed priority basis, with all interrupts fron the Z80 SIO assigned a higher priority than the CTC.

Peripheral support for the 8086 is provided by a mixture of intelligent support chips and combinations of simpler standard logic elements.

The intelligent support chips include:

1. A Western Digital WD2797-02 Floppy Disk Controller for controlling the MicroFloppy Drive.
2. The Zilog Z80 SIO/2 which interfaces to the RS232C port, receives data from the infra-red keyboard link, generates sound and also provides part of the interrupt structure.
3. The Zilog Z80 CTC which acts as a general system timer, determines the baud rates for the RS232C port and also provides part of the interrupt structure.

Another processor is located within the Keyboard. This is a NEC 7507 4-bit processor. It is employed to perform keyboard scanning, encoding of detected keys into a suitable format for transmission via the infra-red link, and the implementation of a real time clock/calendar.

Memory

The F1 is fitted with a minimum of 256 Kbyte of system RAM (using 64K DRAMs) and is expandable by fitting one of the standard Apricot RAM expansion boards (128K, 256K or 512K RAM Expansion boards) into the F1 Expansion Slot (Initially for the USA market, the F1 is fitted with 512K RAM as standard using 256K DRAM).

The other major area of memory within the machine is the Boot ROMs. The Boot ROMs contain a great deal more code than the original Apricot pc/xi machines. Instead of just containing a bootstrap loader, diagnostics, calculator, and a rudimentary screen handler (as with the original disk-based BIOS machines of the Apricot pc/xi range), the majority of the BIOS has also been incorporated within ROM.

The ROM BIOS includes all the device driver routines for handling the standard hardware devices of the F1 (screen, keyboard, disk drive, parallel port, serial port, etc - see Software Chapter following).

The BIOS also contains the generic Apricot applications interface for communicating with the low level BIOS hardware device driver routines; the extended Control Device Driver, as implemented on the Apricot Portable. The same standardised interface format is currently available on the Apricot pc/xi range of products in a limited form. This will be upgraded on the pc/xi range before the end of the year to match the functionality of the Apricot F1 and Portable products.

Producing a ROM-based BIOS for the F1 has the major advantage over disk-based BIOS machines of not occupying valuable code space in the system RAM which could be otherwise utilised by applications software.

The F1 is fitted with two 16K x 8 bit ROMs to store the ROM-based BIOS (i.e. 32 Kbytes of code space). The board containing the memory is also tracked to take 32K x 8 bit ROMs to allow for future BIOS expansion.

Disk Drives

The F1 incorporates a single integral MicroFloppy Disk Drive. The disk drive slot is located on the front facia of the F1. A disk eject button is provided to ensure swift and easy removal of disks.

The disk drive fitted within the F1 is a Sony double-sided MicroFloppy which uses 3.5 inch disks with a formatted storage capacity of 720 Kbytes of data.

The double-sided MicroFloppy disks contain 80 tracks per side, and are soft sectored with 9 sectors per track and 512 bytes per sector. The software format is a logical derivation of the IBM system 34 format for 8 inch disks and is common to all the MicroFloppy based products in the Apricot range (i.e. both the 70 track single-sided and 80 track double-sided disk drive based machines).

BIOS support is provided within the F1 to allow single-sided 70 track MicroFloppy disks to be read from, written to and formatted within the F1's 80 track double-sided drive.

Prior to transportation, a packing disk should be inserted into the F1s' disk drive as a safety measure. This is necessary to avoid the possibility of excessive vibration causing damage to the disk drive heads.

The F1 can also be easily upgraded into a Winchester based machine, using the Apricot MSD (Mass Storage Device) option. This consists of a pre-formatted 10 Mbyte Winchester Disk Drive, a Winchester Controller Board and a separate Power Supply Unit. The controller board fits into the F1s' Expansion slot. Both the 10 Mbyte drive and Power Supply are positioned externally to the F1.

Support for the Winchester Disk Drive is also incorporated within the standard F1 BIOS to allow the user to instantly have access to the extra storage space.

Display Features

The design of the display circuitry of the Apricot F1 is slightly different from the usual microcomputer display architecture.

The first major difference is that there is no high level CRT controller for generating display timing signals and display address lines. These are instead implemented by a variety of simple 74LS series components.

The second major difference is that there is no hardware differentiation between text and graphics; everything is pixel-based. i.e. A "dot" on the display screen is mapped by a corresponding bit(s) in the display memory.

In other words, it does not matter whether the F1 is displaying text or graphics, the display circuitry treats them both in an identical manner. This feature of the design makes it easier for the programmer to mix text and graphics as required by more and more integrated text and graphics based applications and window orientated operating systems.

The display memory is part of the system RAM and occupies 40.5 Kbytes in the lower 64K. 40 Kbytes are allocated in the system RAM to map out a pixel image of the display screen; the other 512 bytes are used to implement a series of 16-bit addresses which form a pointer to map each display scan line.

The modes, resolutions and display features available to the programmer provided by the display RAM are detailed in the next few paragraphs. The resolutions described match the resolutions of the current ACT colour and monochrome monitors produced for the F1.

The F1 can be configured to drive either a colour or monochrome monitor with the programmer having the choice of displaying either 200 or 256 lines.

The programmer also has one further option, either using an 80 column/640 pixel mode or a 40 column/320 pixel mode.

In the 640 pixel mode, the programmer can display up to 4 colours simultaneously (from a choice of 16) on a colour monitor, or up to 4 levels of greyscale if a monochrome monitor is connected instead.

In the 320 pixel mode, the programmer can display up to 16 colours simultaneously on a colour monitor, or up to 8 levels of greyscale if a monochrome monitor is connected instead.

The 320 pixel/40 column mode is the mode which produces a sensible display output on a standard TV. (The relatively low bandwidth of a TV compared with a video monitor does not generally allow a sharply defined picture to be produced in the 640 pixel modes).

Colour/greyscale selection is provided by a palette. This is a small area of memory-mapped RAM which determines the colour mix/grey levels at the display outputs.

The 200 line modes have been implemented primarily for USA usage and other countries using 60 Hz mains supply frequency. The two 200 line modes as described previously are:

1. 640 x 200 bit-mapped graphics using any 4 colours (or grey levels) from 16.
2. 320 x 200 bit-mapped colour graphics using 16 colours (or 8 grey levels).

The higher resolution 256 line modes are for UK, European and other countries using 50 Hz mains supply frequency and are as follows:

1. 640 x 256 bit-mapped graphics using any four colours (or grey levels) from 16.
2. 320 x 256 bit-mapped colour graphics using 16 colours (or 8 grey levels).

A default font of 128 characters (based within an 8 x 8 pixel cell) is contained in the system ROM. This is designed to be used with the 200 line resolution modes.

Each character is mapped by eight contiguous bytes in the ROM. A second font of 256 characters (7 x 7 characters based within an 8 x 8 pixel cell) is loaded into the system RAM at boot-up. Support in the BIOS also allows other 8 x 8 user-defined fonts to be installed within the system RAM. These can be easily accessed by simply modifying a font pointer.

A second default font of 256 characters (7 x 9 characters based within an 8 x 10 pixel cell) is loaded into the system RAM at boot-up. This is designed to be used with the 256 line display modes. Each character is mapped by ten contiguous bytes in RAM.

Support in the BIOS also allows other 8 x 10 cell user-defined fonts to be installed within the system RAM. These characters can also be easily accessed by simply modifying a font pointer. The 8 x 10 based font is of a greater resolution than the 8 x 8 based font for the 200 line modes but is of an identical 256 character set. The basic difference is in the construction of the characters, with lower case letters generally having longer descenders.

To obtain a sensible and usable "text mode" on both the colour display and monochrome display for existing text based applications, the attribute support by the BIOS is only allowed in "monochrome" on the colour monitor (i.e. any two colours from the possible sixteen) and any two grey levels on a monochrome monitor.

All the standard character attributes are available to the programmer in these two modes. These are produced by direct bit manipulation of the character image in the display RAM. Both normal and reverse video characters are supported with any combination of the following attributes:

1. Underline.
2. Strikethrough.
3. Intensity (simulated by shadow printing).

BIOS support for character attributes are not provided in the multi-colour modes due to the inherent nature of the colour display itself. (The same applies to the modes with more than two grey levels on a monochrome monitor).

Since the only effect an attribute is used for is to differentiate a character(s) from other characters, any of the standard attributes can easily be represented by assigning attributes to a different colour in a multi-colour mode (corresponding to a different shade of grey on a monochrome monitor), instead of the standard "monochrome" method.

Keyboard

The design of the F1 Keyboard is slightly different from the keyboards found on other business micros; being a full function keyboard (92 keys) which is linked to the Systems Unit by infra-red and also incorporates a real time clock calendar (implemented in software).

The key layout is divided into a number of well defined sections. These are, looking from left to right across the key tops:

1. The QWERTY section which includes cursor, scroll and general editing keys. This is an identical layout to the one found on the Apricot pc/xi range of computers.
2. A calculator keypad.
3. 10 general/fixed fuction keys.

These keys are square in design and feature a slightly sculptured keytop to ensure accurate user action.

Four machine function keys are located above the keyswitch array and are of an entirely different design. They are slightly recessed to avoid inadvertent user action.

The Keyboard is designed to be used with its spring-loaded feet extended. Buttons for releasing the feet from their storage position are provided on the side of the Keyboard.

The major advantage of using the infra-red link for transmission of keyboard data is that the user is free to site his Keyboard in the general vicinity of the Systems Unit but not necessarily directly in front of it. (Maximum practical distance for using the infra-red Keyboard is specified at up to 2 metres away from the Systems Unit).

To avoid the possibility of interference in multiple machine environments, a "light-pipe" is available for linking the Keyboard and Systems Unit together. This is a section of fibre optic cable which directs the infra-red keyboard transmissions to the receiver circuits of the parent Systems Unit.

To improve system reliability and ensure that the BIOS does not misinterpret data transmitted from the Keyboard, the keycode data is encoded using Hamming codes prior to transmission. This is an error correction/detection encoding technique which allows the BIOS to correct and detect errors in the transmitted key data.

The keycodes are transmitted in serial packets of data, each packet consisting of 32 bits. The information contained in the transmission packet signifies the X-Y co-ordinate of the pressed key and the key status. The key status identifies whether the key pressed is:

1. Shifted (SHIFT key + key pressed).
2. A control key sequence (CONTROL + key pressed).
3. In Auto-repeat mode (key was the last key to be transmitted and is being held down).

The use of keycodes rather than using the ASCII equivalent to represent the key(s) makes it particularly easy for the programmer to redefine the keycode. Support in the BIOS is provided to allow this to be done by simply loading a new keyboard table into RAM and modifying a pointer to point to it. A default keyboard table is stored in ROM.

Not all keys can be reassigned by the applications programmer. Certain of the keys are designed to perform specific functions and are therefore masked off by the BIOS and processed in an entirely different manner. These include the TIME/DATE key and the four button keys RESET, REPEAT RATE, SET TIME and KB LOCK.

The TIME/DATE key causes the time and date information generated by the real time clock/calendar software routines within the Keyboard to be transmitted to the Systems Unit.

This is used by the ROM BIOS to update the BIOS internal clock, (as used by MS-DOS for its time and date stamp). The time and date data is supplied to the Systems Unit in 15 separate contiguous data packets following the TIME/DATE keycode packet.

The key also serves another function at machine switch on, where it initiates the boot loading sequence, if a bootable disk is within the disk drive.

The function of the RESET key is self-explanatory, being the system reset key. It generates a hardware reset in the Systems Unit and must be held down for approximately one second before it functions. The delay is implemented to prevent the user accidentally resetting the system.

The REPEAT RATE key is a toggle switch which allows the user to set the auto-repeat rate of the keys to either one of two values; a fast or a slow rate. (The repeat rate is the rate the keyboard transmits the keycode to the Systems Unit, when a key is held down).

The SET TIME key is used to adjust the real time clock/calendar software within the keyboard. It can be actioned by the user anytime (before or after the system boot). Pressing the key displays a prompt on the 25th line of the display in the following format:

HH:MM DD/MM/YY

The user resets the time and date within the keyboard by typing in numerical values only using the numeric keypad (e.g. typing 1000011285 sets the keyboard clock to 10 am, 1st Dec 1985).

The key does not send the updated time and date information to the ROM BIOS. This function is actioned by the TIME/DATE key as described above.

The KB LOCK key is a toggle key which enables the user to deactivate the effect of all keys apart from RESET, SET TIME, TIME/DATE and KB LOCK itself, (i.e. it locks out the keyboard). Pressing the key again informs the BIOS to restore action to all the keys.

Another special key function on the Keyboard is the CALC key in the shifted mode (SHIFT key + F4) which can be redefined by applications software if required, but should generally not be reconfigured. The key sequence can be used prior to boot and can be also made available during applications or at the operating system level, to initiate the BIOS calculator software.

The calculator display appears on the 25th line of the display screen. The calculator keys are formed by:

1. The numeric keypad (1 to 9, the mathematical function keys, decimal point and ENTER).
2. The CLEAR key.
3. The function keys, STORE, RECALL, M+, M-.
4. The CALC key.

After boot an extra calculator key is available to the user. This is the function key, SEND (obtained by CONTROL + F5). It enables the user to send the results or operands of a calculation to the cursor position on the screen.

The Keyboard is powered by four AA batteries, which are located behind a panel on the base of the Unit. These provide enough power to keep the Keyboard operational (under normal everyday usage) for approximately 6 months.

To cater for custom keyboard layouts (Dvorak, the French style AZERTY format, or any other foreign language layout), the key tops have been designed to be easily removed and repositioned. Applying slight leverage underneath a keytop releases it from its normal location.

Because the majority of the keyboard is software configurable, the programmer can reassign the keyboard to match a different layout simply by installing a new keyboard table, as described previously.

Printer Support

The F1 has two ports available for connecting printers; a Centronics port for parallel printers and an RS232C port which can be used for serial printers.

The Centronics port connections support two of the common handshake signals normally required/supplied on the majority of parallel interface printers:

1. Data Strobe
2. Busy

A description of the facilities provided by the RS232C port for serial printers and various communications device is detailed below.

Communications

A sophisticated RS232C communications port is provided as standard equipment for general purpose communications (via acoustic couplers, modems, direct connection to other micros, etc). It can also be configured for driving various printing devices (serial line printers, plotters, typesetters, etc).

The port can be programmed to operate in both asynchronous and synchronous modes, with the programmer having independent control over transmit and receive baud rates. These can be either set to the same value or set to operate with different rates for transmit and receive as required.

The baud rates can be selected under software control to be driven by an internal timer circuit at any of the more commonly used values (from 0 to 9.6 Kbaud) for general purpose communications.

Alternatively, a software switch enables the baud rates to be set by external equipment instead of the internal timer.

The programmer is able to choose from a variety of synchronous modes. These include the bit oriented modes HDLC and SDLC, and the byte oriented modes, Monosync and Bisync.

The control and timing signals available at the RS322 output (formed by a standard 25-pin D-type female connector) are as follows:

1. RTS (Request To Send)
2. CTS (Clear To Send)
3. DSR (Data Set Ready)
4. DTR (Data Terminal Ready)
5. DCD (Data Carrier Detect)
6. TxCK (Transmit Clock)
7. RxCK (Receive Clock)

Two supply outputs (+12V/—12V) are also available on the connector. These are primarily for use by the Apricot Point 7 network.

Other communications facilities available to the F1 are provided by optional Expansion Boards. The F1 is hardware compatible with the Apricot integral Modem and the Apricot LAN card.

The applications-driven Apricot integral Modem provides the user with the facility to communicate over the telephone network via the F1.

The Modem is a frequency shift keyed (FSK) Modem conforming to CCITT V21 (300 bps full duplex) and CCITT V23 (1200/75 bps full duplex) standards. It has autoanswer capability conforming to CCITT V25 standard, and also incorporates an integral loop disconnect (pulse) autodialler.

Typical applications for the Apricot F1 complete with Modem are:

1. Emulation of various computer terminals which are used for communicating to mainframes and minicomputers.
2. Access public and private databases.
3. Transfer files and data between the F1 and any other micro or computer with asynchronous modem facilities available.

The Apricot LAN card with the appropriate network software allows the F1 to be linked into the Apricot Point 32 network and function as a user network station. This provides the F1 with the facility to access all the allocated resources (printers, file space, etc) provided by this powerful local area network.

Expansion

A single Expansion Slot has been designed into the F1 to cater for any single optional expansion cards the user may require. A plastic expansion plate is also located on the rear panel. This can easily be removed to allow external equipment to be connected to the expansion board with a minimum amount of modification to the machine.

A high degree of compatibilty has been maintained in the design of the Expansion Slot with the other products within the Apricot pc/xi range of computers. This is such that all existing ACT Expansion boards (Winchester Controller, Modem, RAM cards, etc) can be used within the F1 Expansion Slot without any modification to the Expansion Board hardware.

The philosophy for using multiple Expansion boards is different to the one originally adopted on the Apricot pc/xi range of machines.

The user has the option to link the F1 expansion bus connector (on the side of the Systems Unit) to a separately powered Expansion Unit fitted with multiple Expansion Slots. This user is then able to expand the facilities of the F1 using the special features of the Expansion Unit.

The Expansion Unit is responsible for re-powering the Expansion bus to meet the drive capability of multiple Expansion Slots and also for providing a sensible interrupt structure.

Specification

Processor:	Intel 8086 running at 4.67 MHz.
Memory:	256 Kbyte System RAM.
	32 Kbyte of Boot ROM (expandable to 64 Kbyte).
Disk:	Double-sided MicroFloppy disk drive capable of being used with either 80 track double-sided (720 Kbytes) or 70 track single-sided MicroFloppy disks (315 Kbytes).
Printer Support:	Centronics port and RS232C port.
Comms. Port:	RS232C port capable of being driven in either asynchronous or synchronous modes (Bisync, Monosync, HDLC or SDLC) with selectable baud rates (internally 50 to 9.6 Kbaud; or externally set by data communications equipment).
Expansion:	One Apricot pc/xi compatible expansion slot + one Expansion bus connector for linking in an optional Expansion Unit.
Keyboard:	Full function "soft" keyboard incorporating QWERTY layout, calculator keypad, four machine specific function keys, and a bank of ten "fixed/general" function keys. Linked to the Systems Unit by infra-red. (Optional light-pipe connection for multi-machine environments).
Sound:	Programmable tone/noise generator + integral loudspeaker.

Display Features:

Logic to drive either a colour monitor or a monochrome display in the following modes:

1) 640 x 200 resolution bit-mapped graphics using any 4 colours from 16 (c.f. 4 grey levels for the monochrome monitor).
2) 640 x 256 resolution bit-mapped graphics using any 4 colours from 16 (c.f. 4 grey levels for the monochrome monitor).
3) 320 x 200 resolution bit-mapped graphics using up to 16 colours (c.f. up to 16 grey levels for the monochrome monitor).
4) 320 x 256 resolution bit-mapped graphics using up to 16 colours (c.f. up to 16 grey levels for the monochrome monitor).

Default ROM based character font of 128 characters. Alphanumeric characters based within a 7 x 7 pixel matrix and contained in an 8 x 8 cell for 200 line modes. Default 8 x 8 RAM based character font of 256 characters as above, for 200 line modes.

Default RAM based character font of 256 characters. Alphanumeric characters based within a 7 x 9 pixel matrix and contained in an 8 x 10 cell for 256 line modes.

"Soft" font capability.

Software character attributes in "monochrome" modes;

1) Reverse
2) Underline
3) Strikethrough
4) Intensity

Display Outputs:

1. 9-pin D-type Male - Output for either a colour or a monochrome monitor.
2. Phono jack socket - Output for a composite monochrome monitor.
3. 5-pin Molex (located internally) - Output for a standard TV via optional modulator.

Dimensions:	Systems Unit - Length: 16.5 inches (420 mm) Width: 8.7 inches (221 mm) Height: 6.3 inches (160 mm) Keyboard - Length: 17.7 inches (450 mm) Width: 6.6 inches (167 mm) Height: 1.1 inches (28.5 mm)
Weight:	Systems Unit - 9.6 lbs (4.35 kg) Keyboard - 2.9 lbs (1.32 kg)
Power Supply:	Either 240V or 110V operation (selected by an internal link).
Current consumption:	Approximately 600mA - 240V Approximately 1.2A - 110V
Approvals: (pending)	UL - 114 CSA - C22.2 (No. 154 1983) BEAB - BS415 FCC - Class B, Part 15, Subpart J

Software Overview 1.2

Contents

Introduction

Details

Applications Interface
Operating Systems Interface
MS-DOS
ROM BIOS

Introduction

The control software for the F1 consists of three basic modules; a standard proprietary Disk Operating System (normally MS-DOS) and two BIOS modules; one resident in ROM, the other loaded into RAM at the same time as the operating system. (The BIOS is an acronym for Basic Input Output System).

The ROM based BIOS (hence termed ROM BIOS) consists of a number of basic hardware device drivers (screen, keyboard, disk, etc). These are responsible for controlling all the standard hardware devices within the computer (and also some of the Apricot optional add-on devices, e.g. the Apricot 10 Mbyte MSD).

The drivers in the ROM BIOS are not the only software device drivers supplied with the machine; three other loadable device drivers are also provided as standard software on the release disks, together with a graphics software interface (GSX).

The first loadable device driver is an optional user facility for implementing a RAM disk (RAMDISK.SYS). The user is able to allocate 64K portions of the system RAM to simulate a floppy disk (i.e. a RAM Disk) to greatly enhance system performance.

The format of the entry in the CONFIG.SYS file for this installable device driver requires one argument as detailed below:

DEVICE=RAMDISK.SYS /n

where n represents the number of 64K portions allocated.

Note: A space must be inserted between the RAMDISK.SYS and the backslash.

The system software views the RAM Disk as an extra disk drive and allocates the next free drive designation to it. (i.e. On a single drive system, this would be drive B).

Generally, implementation of the RAM Disk is only feasible on systems employing more than 256 Kbytes of system RAM.

The second loadable device driver is the modem driver (MODEMAPR.SYS). This is also an optional loadable device driver. It provides the programmer with the necessary tools to integrate the communication facilities of the optional ACT integral modem into an application.

The third optional loadable device driver is the Mouse device driver (MOUSE.SYS). This is linked into the system to handle all data transmissions from either the ACT infra-red mouse or the MicroSoft serial mouse. It forms an integral part of the graphics software and is also available for use with other applications.

The fourth device driver is the GSX graphics driver. It differs from the other device drivers described above, since it does need to be installed at boot-up. (All the other drivers described above use the MS-DOS facility for installing loadable device drivers by appending an entry to the CONFIG.SYS file). The GSX graphics interface incorporates its own command file (GRAPHICS.EXE) for installing the software.

The function of the GSX graphics interface is to provide the programmer with a machine independent graphics interface.

Details

Applications Interface

On nearly all microcomputers, the applications programmer has a number of choices for integrating his software into the machine's environment. He can do this in any combination of three possible ways as described below:

1. Using the facilities of MS-DOS.
2. Linking into either the ROM BIOS or other device driver routines.
3. Directly accessing the hardware.

MS-DOS

MS-DOS provides the programmer with a high level machine independent interface for applications programs. It allows programs to run on dissimilar machines (e.g. Apricots, IBM PCs, etc), providing no other machine specific features are accessed.

It also permits the programmer to create installable device drivers at the DOS interface level in a consistent manner. These drivers can either define a new device type to be used on an Apricot (e.g. the generic Modem driver, version 2.0 of MODEMAPR.SYS), or replace an existing driver (e.g. keyboard, screen driver, etc).

ROM BIOS

The way the applications programmer links into the Apricot ROM BIOS is via a simple interface, the Apricot Control Device. This provides a standardised method of accessing the BIOS routines and is adopted in the same generic format on all Apricot computers. (It is currently supplied in limited form on the pc/xi range of machines, but is soon to be upgraded into the same specification as found on the F1 and Portable).

The Control Device allows the programmer to control basic low-level machine functions without having to resort to accessing the hardware.

If a programmer uses this interface for all features and facilities not available through MS-DOS, it will allow him to produce "portable" applications which will run on all members within the Apricot range of computers (Apricot pc/xis, F1s and Portables).

The Control Device thus provides the programmer with a low level machine independent interface to application programs, which is *compatible* across the range of Apricot microcomputers.

The purpose of the Control Device is to hide the differences in the hardware between various models within the Apricot family. This does not mean that the application writer cannot use the special features which are implemented on one machine but are not available on another.

Inherent in the configuration parameters is data to enable the application programmer to identify which Apricot micro his application software is being run on. The writer can modify his software to use any special feature by first determining the machine and then tailoring the routines accordingly.

Hardware

Direct accesses to the hardware will not produce the desirable goal of machine independent code for the applications programmer.

The hardware of the Apricot micros is substantially diverse and port addressing significantly different. Code which writes directly to the hardware will require translation and rewrites for each product within the Apricot range of computers. This will of course make the application totally machine specific.

One area where the BIOS does actually mask out the differences in the hardware is in interrupt support for expansion cards. Even though the interrupt lines on the expansion bus are all wired to the same pins on the expansion board connector throughout the whole range of Apricots, the associated interrupt pointers differ from machine to machine.

To make it easier for the third party hardware vendor to produce compatible boards for the whole range, a software interrupt has been reserved which allows the programmer to set up his interrupt handler vectors and relate them to the physical interrupt line rather than the hardware interrupt pointer.

It will not always be possible for all application programmers to totally ignore the hardware and only use a combination of MS-DOS and the Control Device. Accessing the hardware will be necessary if certain features the programmer wishes to use are not available through MS-DOS or the BIOS (e.g. synchronous communications support via the RS232 port).

Direct accessing to the hardware can cause a few problems, generally associated with contention arising between the BIOS and the application when both are accessing hardware registers which are write-only. (The application and the BIOS would not normally know what has been set up by the other's software, and therefore could overwrite each other's code). This contention invariably results in the machine crashing.

To alleviate this potential problem, the BIOS maintains copies of certain write-only hardware registers which may be of use to the application programmer. These are stored in RAM and accesible to both the BIOS and the application.

The BIOS only changes bits within the hardware registers which are of interest to the BIOS and always makes a copy of any changes in the copy register. In order to avoid contention, the application should always adopt the same procedure.

Graphics

One of the current growth areas in applications software is in the use of graphics displays. Again, as with standard machine functions, one of the most desirable goal for an applications writer is to be able to produce graphics software that is portable across a whole range of machines.

In order to do this, the applications writer requires a consistent software interface for graphics functions. This is provided by the Digital Research module GDOS which is part of the Apricot implementation of GSX.

The GDOS is a machine independent applications interface to graphics functions and is implemented on all Apricot micros.

It provides the programmer with a standard set of primitive graphics operations enabling him, by using a simple calling procedure to:

1. Draw lines, arcs, pie slices, bars and circles of various styles, and colours.
2. Place text on the screen.
3. Plot points.
4. Fill polygon areas in various styles and colours.
5. Program the colour palette to alter the colours available on a colour monitor.
6. Interpret user input from the keyboard and mouse.
7. Configure the machine to match the desired resolution and colour mode, (e.g. 4 colour graphics on a colour monitor or lower
   resolution 16 colour graphics).

Examples of programs which run under GSX are the utility programs contained within Activity, ACT Sketch and ACT Diary.

The GSX system is based on two modules, the GDOS module from Digital Research and the GIOS (Graphics Input Output System), written by ACT.

The GIOS is the low level hardware interface which forms the bridge between the GDOS and the display hardware. This differs from machine to machine (as does the display hardware) and is not accessible to the programmer.

Different versions of the GIOS have been provided on the Portable to use the various display capabilities. GIOSs have been created to support a colour monitor in the following graphics configurations:

1. 640 x 200 line graphics using 4 colours.
2. 640 x 256 line graphics using 4 colours.
3. 320 x 200 line graphics using 16 colours.
4. 320 x 256 line graphics using 16 colours.

The programmer selects the appropriate GIOS to match his requirements by using a command available through GDOS.

Operating Systems Interface

MS-DOS does not make requests for services by communicating directly with the ROM BIOS. All requests are directed via the RAM BIOS to the Apricot Control Device. The RAM BIOS handles all communications between MS-DOS and the Control Device. It interprets:

1. MS-DOS function requests and translates them into calls to the generic Control Device.
2. Status messages returned from the Control Device and translates them into MS-DOS format.

Using a RAM based BIOS for this translation makes it particularly easy for other operating systems to use the routines within the ROM BIOS.

The only major function a different operating system manufacturer has to do to link into the machine is, to write a different RAM BIOS which performs the same function as the MS-DOS version and load it in together with his operating system at boot-up. This cuts out the time consuming exercise for the operating system manufacturer of having to create his own set of device drivers.

All the necessary details to allow other operating systems to produce generic boot disks (as supplied by ACT for MS-DOS) so that they are compatible across the different products in the range of Apricot microcomputers (e.g. Portables, pcs and xis upgraded with the ROM BIOS) are included in later chapters.

MS-DOS

The features of Microsoft's MS-DOS are widely known, currently being the most widely used 16-bit microcomputer operating system.

The initial release of software supplied with the Apricot Portable is version 2.11. This will inevitably be upgraded to MS-DOS 3.05/3.06 in the near future, which features "hooks" for linking into the MS-NET networking software. On its own without the MS-NET module, DOS 3.06 is virtually identical in functionality to MS-DOS 2.11.

MS-DOS 2.11 is an extension of the widely used MS-DOS 2.0 which hailed the introduction of the enhanced operating system features of installable device drivers and tree-structured directory support. The main difference between these two versions is that DOS 2.11 provides support for international languages and uses 8-bit character codes in files instead of 7-bit.

Different versions of DOS 2.11 are available complete with MS-DOS utilities, which support foreign languages such as French and German.

ROM BIOS

The ROM BIOS has been discussed in detail in the Applications Interface section above with reference to the function of the Control Device. Instead of repeating this information, a brief summary of its salient points are detailed below. This is then followed by an introduction to the standard device drivers included within the ROM BIOS.

Besides consisting of the series of basic hardware device drivers, the ROM BIOS also contains the generic applications interface, termed the Apricot Control Device. This is implemented on all Apricot machines, (currently as a limited sub-set on the Apricot pc and xi machines but these machines are soon to be upgraded to the same interface specification as the F1 and Portable).

The routines in the ROM BIOS are accessible to the application programmer via the control device interface. It provides the application programmer with an extremely easy and efficient way to access various low level routines, not normally available using calls to MS-DOS alone, thus allowing the programmer greater control of machine functions.

The programmer can access the Control Device in one of two ways.

The first method is designed to suit assembly language programmers and is similar to a MS-DOS function request. The programmer loads the 8086 registers with:

1. Information to specify the device to be accessed (e.g. Keyboard driver).
2. A command (e.g. initialise)
3. Data as required.

A call is then made to the Control Device by generating the interrupt FCH.

The second method is designed to support higher level languages such as BASIC. The programmer accesses the Control Device by assembling a series of parameters (specifying the device, command and data as required) and passing them onto the stack by performing a far call to 0600H.

Drivers

The ROM BIOS contains the following standard device drivers, listed below. The actual function of each driver is indicated by their titles.

1. Keyboard Driver
2. Screen Driver
3. Disk Driver
4. Parallel Port Driver
5. RS232 Driver
6. Clock Driver
7. Winchester Driver

Keyboard Driver

This routine receives all data transmitted to the machine via the infra-red input. This includes keyboard and mouse data.

Decoded mouse data is not handled by the keyboard driver. It is immediately passed onto another routine via an interrupt. The mouse data handling routine may be the loadable mouse device driver, or any other routine installed by the application writer.

Keyboard data is always initially analysed for any special key depressions such as TIME/DATE, KB LOCK, SET TIME, REPEAT RATE. These keys are filtered off and sent off to the appropriate ROM BIOS routine to action a specific user function. They are therefore not accessible to the applications writer.

All other keys are converted to an Apricot compatible keycode (termed a downcode), which is normally used to select an entry from a keyboard table. The selected entry is then usually passed to MS-DOS via an 80 byte queue.

The keyboard table occupies a minimum of 1K of memory space and can be either the default keyboard table in ROM or any other keyboard table loaded into RAM by the programmer. The use of a software keyboard table allows the programmer to translate a user key depression to any code or sequence of codes as required.

A simple mechanism enables the programmer to specify the keyboard table in use. This is achieved by modifying a pointer (the active key table pointer) to point to the start of the desired table.

A KEYEDIT utility is supplied with the system software to allow the user/programmer to either create new keyboard tables or modify an existing one.

The format of the keyboard table used on the F1 is compatible with the format of the key tables used on all other Apricot computers (Portables, pcs, xis, etc). It consists of four sections.

The first three sections contain the entries for keys in normal mode (single keystroke), shifted mode (key + SHIFT key), control mode (key + CONTROL key). The last section defines an area of string keys. This section of the table is accessed by programming the entries in the three other sections to act as a pointer into the string area.

Facilities are provided within the Control device interface to alter the way the driver handles the keycodes. The programmer has various options available, for manipulating keycode data such as:

1. Handling downcodes directly, missing out the translation process provided by the keyboard table.
2. Analysing and extracting keycodes from the 80 byte queue, and processing the data as required.
3. Placing data into the queue.
4. Checking driver status, sounding the bell, etc.

Interrupt support is also available to enable the application programmer to vector off keycode data to his own routines, as required.

Screen Driver

The features of the Screen Driver provide the applications programmer with comprehensive of the available display options (colour monitor, monochrome monitor, standard domestic TV, etc).

The basis for controlling the displays is via ESCape sequence support and calls to the Control Device.

The screen modes available through the driver are as follows:

1. Standard 80 column by 25 row character based display on either the colour or monochrome monitor (using any two colours from 16 on the colour display or any two grey levels from 16 on the monochrome monitor). The following attributes are also available in normal or reverse video:

   Underline
   Strikethrough
   Intensity

   This is termed the *Apricot compatible* mode, since it is available in the same format on all current machines.

2. 80 column by 25 row character based display using any four colours/grey levels from 16. Standard attributes not supported. Attributes generated by varying the foreground and background colour/grey level of the characters.

3. 40 column by 25 row character based display using any four colours/grey levels from 16. Standard attributes not supported. Attributes generated by varying the foreground and background colour/grey level of the characters. Suitable for driving a standard domestic TV.

The displays can be driven in one of two different scan line resolutions; 200 or 256.

These are designed to match the display resolutions available on an ACT colour monitor when it is connected to either of the two mains supply input frequencies used in the majority of countries throughout the world.

The 200 line mode is primarily for use in the USA and other countries using 60 Hz mains supply lines. The 256 line mode is for use in countries using 50 Hz mains supply input (UK, European, etc).

Two fonts with 256 identical characters per font are normally downloaded into the system RAM at boot-up.

One of the fonts is based upon an 8 x 8 character cell and is available for use on display monitors running in 200 line mode. The second font is based upon an 8 x 10 character cell and is for use on display monitors set for the 256 line mode. The screen driver performs automatic selection of the correct font according to the line resolution mode selected by the application programmer.

A FONTEDIT utility is supplied with the system software to allow the user/programmer to either create new character fonts or modify an existing one.

The programmer can also dynamically change from font to font during run-time (if required). This is achieved by simply modifying a pair of font pointers, which specify the start of the currently active font.

A comprehensive set of ESCape sequences are inherent within the screen driver including a sub-set of the ANSI standards. These provide the programmer with:

1. Control of screen character attributes such as intensity, underline, etc in the Apricot compatible mode.
2. Colour/grey level selection. This includes; background and foreground screen colour/grey level in Apricot compatible mode; background screen colour/grey level plus background and foreground colours/grey levels on a per character basis in the four colour/grey level modes; independent programming of the palette for other application usage.
3. Cursor control routines.
4. Word Processing primitives, such as Insert line, Delete line, etc.
5. Facilities for changing the screen environment, e.g. to 80 columns, 4 colours etc.
6. Windowing functions.
7. Support for hard copy.

To make it easy for the programmer to build character images on the bit-mapped display, the driver supports a virtual screen image located in RAM.

Each character on the virtual screen is represented by a single word. The lower byte is the ASCII character code; the upper byte either signifies the character attributes or if a multi-colour/grey level mode, the foreground and background colours for the character.

The programmer accesses the virtual screen via the control device interface. He can use it to build up character images in the background, and then command the driver to repaint the image on the bit-mapped display from the virtual screen.

Other support features provided by the control device allow the programmer to update individual characters on the screen with the appropriate attribute/colour selection as required.

Disk Driver

This driver has been primarily designed to provide the necessary support for MS-DOS disk operations to the floppy disk drive. The only entry point to the driver is via the Control Device interface.

The driver is configured to support both 70 track single-sided and 80 track double-sided disks, enabling multiple sector reads and writes to either type. It also provides calls for linking in a formatting program.

The applications writer can use the control device calls for checking disk status, and if so desired can perform read and writes to absolute disk sectors instead of using the MS-DOS file structure.

Parallel Port Driver

The parallel port driver is used to drive the Portable's Centronics port, thus providing applications support for sending data to parallel printers and plotters. The only method provided for accessing the driver, is via the Control Device interface.

Facilities provided by the Control Device allow the programmer to:

1. Examine/control the state of the Centronics interface handshaking signals.
2. Send characters to the driver's print buffer (the length of which is 2K bytes).
3. Clear the print buffer of characters, test for space in the buffer, etc.

It also provides a call to re-route characters to a serial printer via the RS232 port.

RS232 Driver

This driver provides various support features to allow the programmer to drive the RS232 port asynchronously. The programmer's method of accessing these routines is via the Control Device interface.

The following features are available through the Control device interface for supporting asynchronous communications:

1. Full duplex operation with variable length buffering available on both the transmit and receive paths (1 to 512 bytes).
2. Control of most of the commonly used transmit and receive baud rates.
3. Selectable parity and stop bits.
4. Control and status monitoring of the modem control lines; DTR, CTS, RTS. DSR and DCD.
5. XON/XOFF flow control.
6. Primitive teletype functions (e.g. automatic transfer of nulls after carriage return, etc).

Other features provided by the driver are:

1. Automatic vectoring of receive data to an installed mouse device driver to enable handling of data from a serial mouse.
2. Interrupt support to allow the hardware (Z80 SIO) to be driven directly by an application without having to resort to writing and reading to port addresses.

Clock Driver

This driver is driven by a hardware timer interrupt which is generated on a regular 20 ms cycle. The routine is responsible for maintaining the clock/calendar for the time and date stamp used by MS-DOS. This is updated when the user presses the TIME/DATE key.

It also handles various timing routines as required by the floppy disk drive, winchester, cursor control routines, printer routines, etc.

The application programmer can use the regular 20 ms cycle to implement his own timer related functions. He can link into the cycle by simply installing his own routines at a location specified by a software interrupt.

Winchester Driver

This driver is provided to support add-on Winchester devices such as the Apricot MSD. As with the floppy disk driver, the main function of the driver is to provide the interface between MS-DOS and the disk drive.

The driver enables up to two Winchester drives to be supported in a single system. The Winchester drives can be of different capacities (e.g. a 5 Mbyte and a 10 Mbyte).

The applications writer can use the control device calls for checking disk status, and if so desired can perform read and writes to absolute disk sectors instead of using the MS-DOS file structure.

There are no routines in the driver for linking in formatting programs.

Options 1.3

Contents

Introduction

Details

Display Options
RAM Expansion Boards
Modem Board
LAN Board
Mouse
Apricot MSD
Expansion Unit

Introduction

The basic configuration of the F1 can be altered by the addition of various options to reflect the differing requirements of the user. These can be broken down into the following categories.

1. Display Options.
2. RAM Expansion Boards.
3. Modem Board.
4. LAN Board.
5. Mouse.
6. Apricot MSD.
7. Expansion Unit

Details

Display Options

These can be further broken down into a number of different types, some of which are available from ACT and others which are available from independent suppliers. The display options available from ACT are:

1. 9 inch F1 Monochrome Monitor.
2. 12 inch F1 Monochrome Monitor.
3. 10 inch F1 Colour Monitor.
4. TV modulator.

Alternatively, the F1 can be used with a standard black and white composite monitor or any other standard IRGB colour monitor set up to match the required line resolution.

The monochrome options are supplied as a monochrome monitor plus a cable which includes an in-line transformer. This plugs into the two pin socket on the back of the F1 to provide the supply for the monitor. The monitor plugs into the 9-pin D-type connector on the back of the F1. It provides the F1 with the facility to display text and graphics using a grey scale.

The colour monitor option also plugs into the 9-way D-type connector located on the back of the machine. It provides the F1 with the facility for displaying text and graphics in colour.

The colour monitor option is a 16-colour IRGB monitor, which is powered directly from the mains supply.

Both the colour and monochrome monitor can be configured to run in either 640 x 200 line display mode (normally when using 60Hz mains supply input frequency) or 640 x 256 line display mode (50 Hz mains supply frequency).

Support in the BIOS allows the applications programmer to display text-based applications in either monochrome (any two colours from 16 on the colour display, any two grey levels on the monochrome monitor), or using a 4 colour mode (any four colours from 16 or any four grey levels).

Graphics support provided by the GSX interface allows the programmer to drive the monitor in either 4 colour mode (using 4 colours/grey levels from 16) or 16 colour mode with reduced resolution (using the full range of colours/grey levels on the monitor).

The F1 can also be connected to a standard domestic television. This requires the optional TV modulator.

The modulator is fitted internally within the F1 and monopolises the Expansion Slot. Connections to the TV are via a standard co-axial cable. TV support in the BIOS is provided by the 40 column modes. Support in GSX is provided by the 320 pixel resolution modes.

RAM Expansion Boards

The RAM Expansion Boards are single board Expansion cards. The boards are available in three different memory sizes, 128 Kbyte, 256 Kbyte and 512 Kbyte. In the F1, the appropriate board can be installed to increase the standard 256 Kbytes to any one of the following values:

1. 384 Kbytes.
2. 512 Kbytes.
3. 768 kbytes.

Modem Board

The Apricot Modem is an integrated hardware and software communications package, which provides the F1 with the facility to transmit and receive data via the Public Switched Telephone Network (PSTN).

The Modem communications package is provided as follows:

1. The Modem hardware, which fits internally within the F1 utilising the Expansion Slot.
2. The Modem device driver software which is supplied as a loadable device driver which is part of the release software.

The Modem is driven via an applications software package interacting with the Modem device driver. This allows the programmer to define the particular service or use, the Apricot F1/integral Modem combination is to be configured for.

Both the hardware and the software device driver have been specifically designed to allow the Apricot/Modem combination to operate as a multi-purpose communicating microcomputer with a vast and diverse range of differing capaabilities, as defined by an applications program.

Typical applications for which the Apricot complete with Modem can be employed are detailed below:

1. Emulation of various computer terminals which are used for communicating to mainframes and minicomputers.
2. Act as an interface to British Telecom's viewdata services. This includes the public viewdata service Prestel, or any of the private viewdata services which are protocol compatible with Prestel. Details of Prestel are widely known. The less known service private viewdata, is operated by large organisations for dissemination of information from a private data base to dealers and clients. e.g. British Leyland's dealer information service.

3. General purpose networking for transferring files and data between the Apricot and any other computer with asynchronous modem facilities available. Communications are not restricted in terms of distance. Both long distances up to thousands of miles (via the public telephone network), or even short distances within the confines of a building (limited local area network capability via a PABX), can be easily accommodated.
4. Function as a repertory dialler. (i.e. A telephone management system which provides automatic dialling of telephone numbers, selected from an internal directory for either voice or data connection).

The F1/Modem combination is not limited in its connection to a telephone network; it can be connected directly to the network or indirectly through the majority of PABXs with loop disconnect dialling facilities.

Connecting the F1 fitted with the Modem into the telephone system is a simple operation. The Modem is fitted with a "flying lead" terminated with a series 600 plug. The F1 can thus be easily connected anywhere on the PSTN by way of a standard series 600 socket.

LAN Board

The Apricot LAN Board is an Expansion Board, which allows the F1 to be linked into the Point 32 local area network as a workstation. This immediately provides the user with access to the resources allocated on the network (large capacity Winchester storage facilities, shared files, shared software, network printers, etc).

To link into the network requires both hardware and a software package (supplied with the network). The hardware consists of the LAN Board and a simple jack plug connection for linking into the network.

The software for the point 32 network is based upon Microsoft's MS-DOS 3.06 plus the MS-NET module. This provides the front end for both the user and the applications programmer.

The lower level communication mechanism for the network is based upon the Corvus Omninet. This is a 1 Mbit/s synchronous bit-oriented (SDLC style) transmission system which employs a carrier-sense multiple access/collision avoidance protocol to ensure the integrity of data transmissions.

A network software device driver (often referred to as the transport layer) is used to link the Microsoft modules to the low level Corvus Omninet System.

Infra-red Mouse

The Mouse for the F1 is identical to the Mouse for the Portable (apart from the colour of the plastics). It has been designed to be used either as a Mouse (by tipping it forward and rolling it along the desk), or as a tracker ball (keeping the Mouse stationary and moving the ball by finger movements).

The mouse is normally employed for cursor movement control and menu selection in graphics environments, but can be used within other applications as required.

A mouse device driver is supplied with the standard release software to allow applications to use the features and facilities of the device. This is an installable device driver which is loaded into the system using the MS-DOS CONFIG.SYS file mechanism.

The Mouse uses infra-red technology in a similar way to the Keyboard. As with the Keyboard, the Mouse can be sited within the vicinity of the F1 but does not necessarily have to be directly in front of it. (The front edge of the mouse must of course point at all times during usage in the general direction of the front of the machine). The maximum practical range of the Mouse is specified at 2.5 metres away from the Systems Unit.

To avoid the possibility of interference in multiple machine environments, a "light pipe" is also available for linking the Mouse and Systems Unit together. This is a section of fibre optic cable (similar to the Keyboard cable) which directs the infra-red Mouse transmissions to the receiver circuits of the parent Systems Unit.

A two-position switch is located on the base of the unit. This should be set to the position towards the rear edge if using the light-pipe and the other posion if not. The function of the switch, is to turn off one of the infra-red transmitting LEDs to conserve battery power.

To improve system reliability and ensure that the BIOS interprets the data transmitted from the Mouse correctly, the Mouse data is encoded using a similar format as used for the Keyboard. This employs a four byte synchronous data transmission format with each data byte encoded with Hamming codes.

Mouse data is transmitted in serial packets of data, with each packet consisting of a 32 bit code sequence. The information contained in the packet signifies the relative movement of the Mouse from it's previous position, and the state of the two Mouse buttons (pressed or not pressed).

Apricot MSD

This device provides the F1 with instant access to a large capacity (10 Mbyte) Winchester Disk. It is supplied as three items; a Winchester Controller Board, a Winchester Disk and a small power supply unit. The Winchester Controller Board plugs into the F1's Expansion Slot, the other two items are mounted externally to the Systems Unit.

The Winchester drive is supplied pre-formatted complete with system tracks. It is configured as a single volume (drive ***A***) to take full advantage of the tree-structured directory features of MS-DOS 2.11 and its future derivatives. (The floppy disk drive in a single Winchester system is automatically re-assigned as drive ***B***)

Support for a Winchester is inherent in the standard ROM BIOS. All the user has to do to use the Winchester is install the components correctly and switch on.

Included in the initialisation routines of the ROM BIOS is a routine which checks for the existence of a Winchester Controller Board. If present, it checks the Winchester Disk Drive to determine it's size.

At the end of the initialisation sequence, the ROM BIOS displays the startup screen. If a Winchester is present, this is slightly modified from the standard display, to include the size of the Winchester Disk.

At the start of the boot sequence, the ROM BIOS first checks the floppy disk drive for a bootable disk. If not present, the machine boots automatically from the Winchester Disk, providing the user with instant access to a large non-volatile storage medium.

Expansion Unit

The F1 Expansion Unit provides the F1 with multiple Expansion Slot capability. It is a separately powered unit and has been designed to be linked into the F1 using the F1 expansion connector. This is accessible by removing a small cover panel on the right hand side of the machine.

The cable extension extends the expansion bus out of the F1 into the Expansion Unit. Here it is bufferd and re-powered to provide sufficient drive capability for two Apricot compatible expansion slots.

Section 2
Hardware Detail

Systems Unit 2.1

Contents

Introduction

Details

Mechanics
Connectors
Front Panel
System Board
Infra-Red Detector Board
Disk Drive Unit
Expansion
Power Supply
Physical dimensions

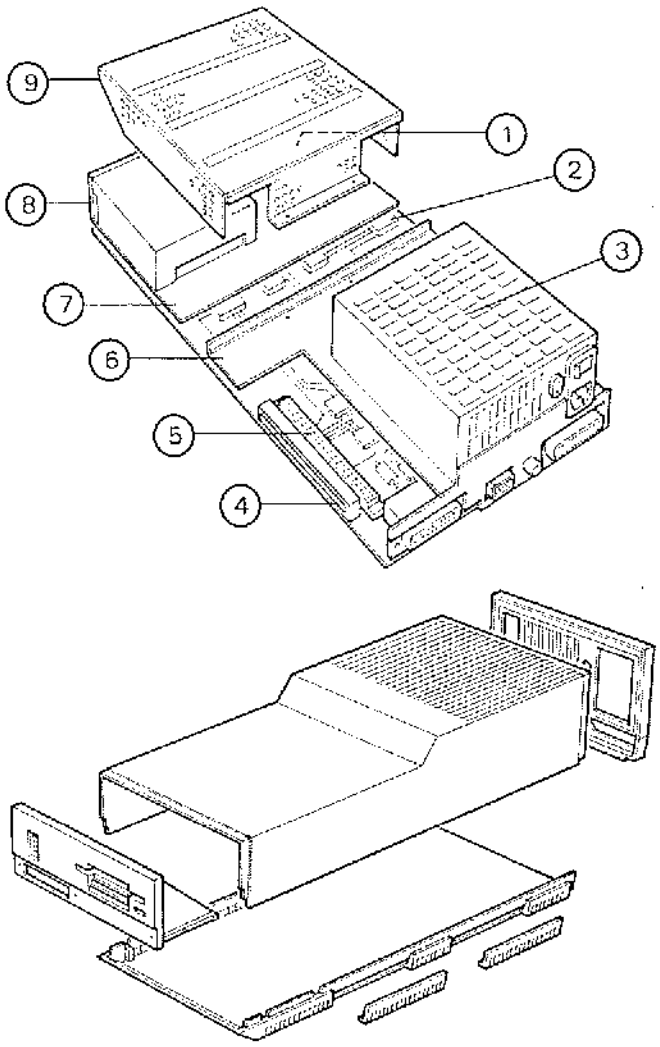
Illustrations

1. Systems Unit
2. Rear Panel Detail
3. Systems Unit Schematic

Introduction

The Systems Unit is the box which houses the the majority of the electronic and electrical components of the F1.

This chapter describes the physical and electrical details of this unit.



1. Infra-Red Detector Board
2. System Board
3. Power Suppy
4. Expansion Connector
5. Expansion Slot
6. Metal Bridge assembly
7. Metal Bridge assembly
8. Disk Drive
9. RF Shielding

Figure 1. Systems Unit

Details

Mechanics

All the electrical and electronic components are contained internally within the plastic case that forms the Systems Unit.

The majority of the processing circuitry is contained on a single printed circuit board; termed the System Board. Infra-red detectors and a small section of logic circuitry are contained on a second board, the Infra-Red Detector Board (see Figure 1).

The System Board lies flat along the bottom of the Systems Unit. On the System Board are the main processor, the system memory, and their associated clocks, plus timing logic and control bus circuitry. Two expansion bus connectors are fitted onto the board for extending the system bus (address, control, and data lines).

Also on the System Board are the controllers and interface circuitry for; the disk drive, various types of display monitor, infra-red input data, a parallel printer, RS232C serial communications and sound generation.

The Infra-Red Detector Board is mounted on a metal bridge above the System Board and encased within a metal box. It is located at the front of the Systems Unit. Photodiode detectors are mounted on the edge of the board, to capture IR transmissions from the Keyboard or Mouse. The detector board translates IR pulses into electrical signals and supplies them to the decoding circuitry on the System Board.

Mounted next to the Infra-Red Detector Board on the same metal bridge above the System Board, is the MicroFloppy Disk Drive Unit and a loudspeaker.

At the rear of the Systems Unit is another metal bridge assembly, which supports the Power Supply Unit.

The disk drive, IR Detector Board and speaker and power supply components are surrounded by shielding. The internal connectors to these components are all located in the middle section of the System Board, accessible between the two bridges.

After the back panel has been removed and internal links to the other components have been disconnected, the System Board can be slid horizontally out of the Systems Unit for servicing.

Cooling is achieved by convection. Heat is generated principally by the Power Supply Unit and the MicroFloppy Disk Drive Unit. Vents are incorporated in the design of the Systems Unit plastics to ensure an adequate dissipation of waste heat without the need for a cooling fan.

Connectors

The connector for the mains power input, and the connectors for standard peripheral units (printers, plotters, external modems, display monitors, etc) are all located on the rear panel of the Systems Unit (see Figure 2).

Viewed from left to right, the peripheral connectors along the Systems Board at the bottom of the rear panel are:

1. An RS232C serial communications port (25-pin female D-type).
2. A Display Unit connector for either an F1 monochrome or an F1 colour Monitor (9-pin male D-type).
3. A Video Jack Plug socket for a composite TV monitor.
4. A Parallel Printer Port (36-way female Centronics connector).

Above the Parallel Printer Port is the mains power input socket (3-pin male) with the power on/off switch above that. The mains fuse is accesed by prising open the hinged panel from the top which protects and surrounds both switch and socket.

To the left of the mains power input is a power input socket (2-pin male) for a 17 Volts A.C. supply. This is used only with the F1 monochrome monitors and provides the means of powering the monitor.

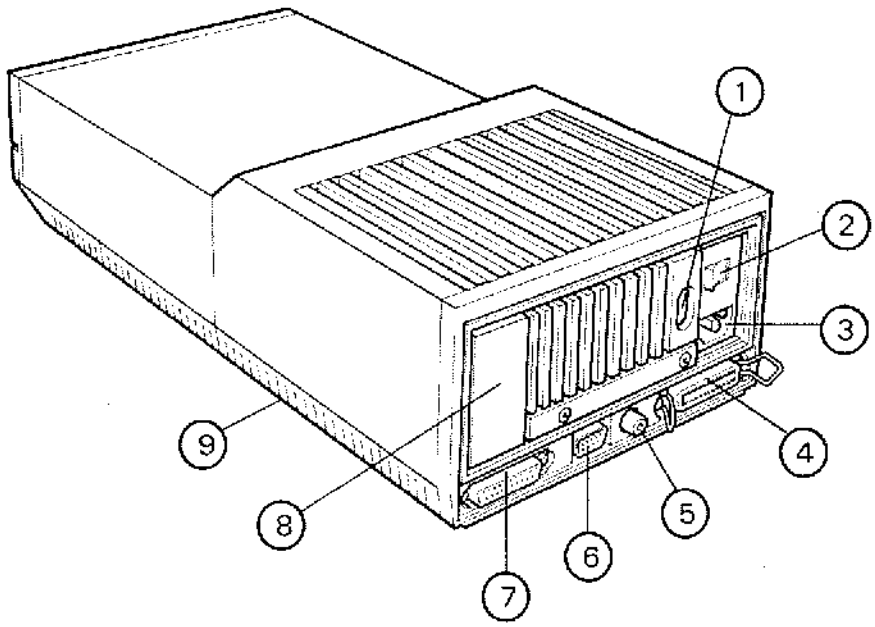
The F1 also has the ability to drive a standard domestic TV but requires an optional modulator which makes use of the internal Expansion Slot.

At the left hand end of the rear panel there is a plastic expansion plate. This can be removed to allow external equipment to be easily connected to Expansion cards fitted into the internal Expansion Slot with the minimum of modification to the rear panel.

An external Expansion Connector is located next to the internal Expansion Slot on the System Board. This is provided for linking in the optional Expansion Unit. Removing a cover panel from the side of the Systems Unit, provides the necessary access to the connector for the Expansion Unit.

Another cover panel is also located on the righthand side of the F1 but closer to the front. It can be removed to allow the control lines from the Floppy Disk Controller on the System Board to be routed to a second MicroFloppy disk drive.

This requires the existing Floppy Disk Controller ribbon cable assembly to be replaced by a daisy-chained cable with two connectors; one for the internal drive and one for the external drive. (Routines are inherent within the BIOS which support a dual disk drive configuration).



1. Power Supply Input—Monochrome Monitor
2. Mains Switch and Fuse assembly
3. Mains Input
4. Centronics Connector
5. Composite Video Connector
6. Colour/Monochrome Display Connector
7. RS-232 Connector
8. Expansion Plate
9. Expansion Cover

Figure 2. Rear Panel Detail.

Front Panel

The front panel of the Systems Unit contains the slot for loading disks into the MicroFloppy Disk Drive, a column of four status display LEDs, and a transparent window through which the majority of infra-red transmissions are captured by a wide-angle lens mounted on the IR detector board.

Behind the transparent window on the IR Detector Board are three photodiodes; one surrounded by the lens, the other two fitted into sockets on the front panel. The sockets are for connecting light pipes from the Keyboard and/or the Mouse.

The right hand socket operates a switch which cuts out the photodiode receiver surrounded by the lens to prevent interference from other infra-red sources in multiple machine environments. The other two receivers are unswitched and are always left switched on.

The status display LEDs are used to indicate:

POWER	Systems Unit power on
CAPS LOCK	CAPS LOCK key active
STOP	STOP key active
DISC	Disk drive in use

System Board

The System Board incorporates the circuitry which perform the processing tasks within the system. Connectors that link the board to the other units within the Systems Unit and to external peripherals are also located on the board.

The major circuit components on the board are:

1. The 8086 processor operating at 4.67 MHz.
2. The Boot PROMs, which contain the system code for bootstrap loading and the BIOS driver routines.
3. The system RAM (256 KBytes of dynamic RAM).
4. A Z80 Serial Input/Output (SIO) controller, which incorporates two independent communication channels, one for RS232 serial communications, the other for infra-red input data and for generating sound.
5. A Z80 Counter/Timer Circuit (CTC), with four independent, programmable counter/timer channels.
6. A WD2797-02 Floppy Disk Controller (FDC).
7. Display controller circuitry.
8. A 16 x 4-bit static RAM, which is used as a programmable Colour Palette.
9. An addressable, 8-bit Latch, which is used as a Control Port.
10. An 8-bit Data Latch for the Parallel Printer Port.
11. A 14 MHz oscillator which forms the fundamental frequency source for all the clock signals on the board.

Other ancilliary circuitry includes; data latches, multiplexers, transceivers, counters, and decoders. These are used as support control circuitry for synchronising signals on the System Bus, refreshing the DRAMs and cycling through the screen refresh addresses, and buffering transfers to external peripherals.

The chapter following provides a detailed desciption of the System Board and of its main functional components.

Infra-Red Detector Board

The Infra-Red Detector Board is located above the System Board behind the transparent window in the front panel of the Systems Unit. It is linked to the System Board by a 4-wire cable assembly. This provides +12V and +5V dc regulated supplies to the board and also carries the decoded IR signal pulses to the System Board.

The Board incorporates three photodiode detectors, an amplifier section, and a timer circuit (see Circuit Diagram in Appendix D). It converts input infra-red pulses into a form suitable for use by the receiver circuits on the System Board.

Two of the photodiodes are fitted behind sockets on the right of the transparent window. These sockets are for connecting light pipes to the Systems Unit.

A third photodiode detector is surrounded by a lens, in order to optically amplify freespace infra-red transmissions.

Fitting a light-pipe into the right hand photo-diode socket operates a switch which switches the diode surrounded by the lens off. This action is necessary to reduce the chance of interference from other infra-red sources in multiple machine environments.

Detected infra-red pulses are converted into electrical pulses by the diodes. To account for the variations in signal strength of the input infra-red pulses, and to prevent saturation of the diodes, the current through the diodes is regulated by an a.g.c amplifier (Q1).

Following conversion to a voltage, the raw input pulses are amplified by a high gain amplifier circuit (Q2, Q3). This produces a pulse output with an amplitude of approximately 2 to 5 V which is then supplied to a 555 timer circuit.

The timer is wired in a non-retriggerable mode to prevent false triggering once a pulse is detected. The timer "squares" up the raw input pulses of 18 to 20 μs duration and converts them into 25 μs duration output pulses. These are supplied to the System Board.

On the System Board, the transmitted signal is separated into timing and data pulses, so that the data signal can be clocked into the Z80 SIO as a standard monosync transmission.

Disk Drive Unit

The MicroFloppy Disk Drive Unit is mounted on a metal chassis above the System Board, behind and slightly on the right of the front panel.

The disk eject button of the Drive Unit fits through the front panel to provide the means for ejecting disks out of the disk drive.

A ribbon cable assembly connects the disk drive to the disk interface on the System Board. A second, 4-way, cable assembly supplies power from the System Board to the Drive Unit.

The MicroFloppy Disk Drives use 80-track double sided 3.5 inch MicroFloppy disks with a total storage capacity of 720 Kbytes of formatted data. The BIOS is also configured to allow the user to read, write and format 70-track single sided 3.5 inch MicroFloppies.

The DISC indicator on the front panel of the F1 is connected to the control line which controls the loading of the disk drive heads. The indicator is illuminated every time a signal is sent to load the heads and remains so until the disk drive heads are unloaded.

Expansion

Two Expansion connectors are provided for extending the System Bus (address, control, and data lines). One of these is an Apricot compatible Expansion slot. This is located internally close to the Power Supply Unit and is designed to take any of the ACT Expansion Boards.

The second is the Expansion Connector for linking in the optional Expansion Unit, as previously discussed. This is located behind a snap-fit cover on the right hand-side of the Systems Unit.

The internal Expansion Slot is a 64-way connector (DIN 41612 2 by 32-way female with a type B housing). The signal lines connected to the Expansion Slot are 95% functionally compatible with all other machines in the current range of Apricot microcomputers. (The main area of difference is that there are no DMA facilities available on the F1).

A removable Expansion cover panel is located on the rear panel of the Systems Unit to allow any Expansion Boards fitted into the Slot to be linked to external equipment as required.

The external Expansion Connector (60-way male IDC) is positioned on the right side of the Systems Unit. It is designed for linking in the optional Expansion Unit. This unit provides the user with multiple Expansion Slot capability. The unit has its own power supply and repowers and buffers the expansion bus to drive four Expansion slots. The Expansion Unit is also supplied with a —12 Volt supply line from the System Board to power a cooling fan.

Power Supply

General

The mains power input is connected to the Systems Unit via the 3-pin male connector on the rear panel. The mains supply is fed through a line filter via the input fuse and mains switch before being supplied to the rectifying and regulating circuitry of the Power Supply Unit (PSU).

The PSU rectifies mains supplies of either 110 Volts or 240 Volts AC. The unit is reconfigured to the appropriate mains voltage by connecting an internal jump lead across the appropriate pins within the PSU.

The PSU is of switched mode design, providing regulated outputs of +12V, +5V and —12V for use by both the System Board and all the other units within the Systems Unit. Other units are not fed directly with the dc supplies from the PSU, but are supplied via the System Board wiring and various cable assemblies. This includes the Disk Drive Unit, the Infra-Red Detector Board, and any expansion boards that are fitted to the internal Expansion Slot.

All other external devices, except the F1 monochrome monitor, are normally supplied with mains directly to their appropriate input.

The F1 monochrome monitor is powered by a 17V ac supply which is sourced by the transformer supplied with the monitor. This plugs into the two-pin connector on the rear panel.

The 17V supply is routed through to the System Board wiring via the PSU dc distribution cable assembly. It is rectified and switched under relay control through to the 9-pin display connector on the rear panel. Switching the mains supply to the F1 on and off switches the 17V supply to the monitor on and off.

All power supply components are housed in a shielded case. A fuse is located in the mains input line, within the PSU. This is in addition to the user accessible mains input fuse within the mains switch housing.

DC Supply Distribution

The regulated outputs from the PSU are supplied to the System Board via a 7-wire cable assembly, which is terminated at both ends in Molex connectors.

The PSU provides two separate regulated supplies of +5V dc, a single regulated supply of + 12V DC, and another single regulated supply of —12V DC. The 17V ac for the F1 black and white monitor is also routed through the PSU.

The maximum current ratings for the various supplies are detailed below, in the order in which they are supplied to the Molex connector on the System Board (starting nearest the centre of the Board).

1. + 5V supply 3.2A
2. Common ground
3. + 12V supply 1.0A
4. Common ground
5. — 12V supply 0.2A
6. 17 Volts ac
7. 17 Volts ac

Distribution of the supplies from the System Board to the other areas are via the board wiring to the appropriate connector.

The System Board provides:

1. + 12V and +5V to the Infra-Red Detector Board, via a Molex connector and 4-wire cable assembly.
2. + 12V, +5V and —12V to the internal Expansion Slot. (Units connected to the external Expansion Connector must supply their own power; the —12V output is used to power the Expansion Unit fan.)
3. + 12V and +5V to the disk drive, via a Molex connector and 4-wire cable assembly.

Fuse rating

The rating of the fuse on the rear panel of the Systems Unit is as detailed below.

240V mains input - 2A, 5 x 20 mm fast action.
115V mains input - 2A, 5 x 20 mm fast action.

Physical Dimensions

Height:	6.3 inches	(160 mm)
Width:	8.7 inches	(225 mm)
Depth:	16.5 inches	(420 mm)
Weight:	9.6 lbs	(4.35 kg)

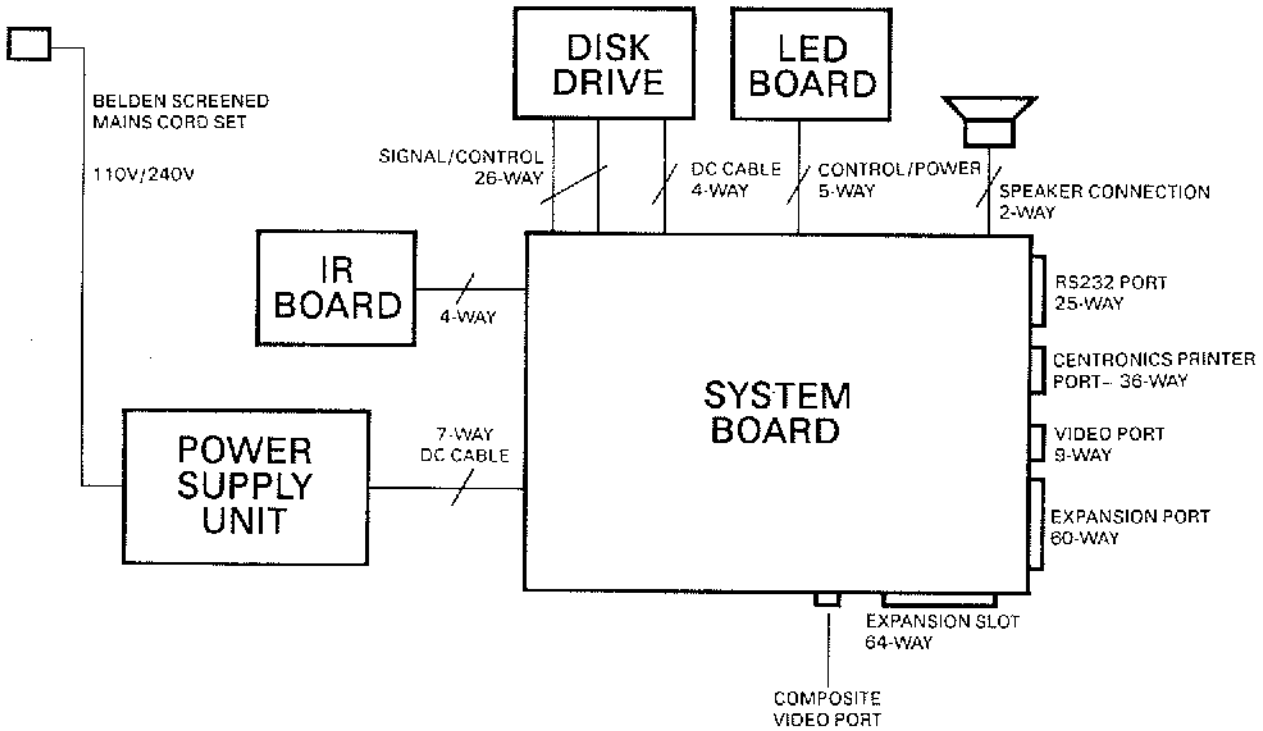


Figure 3. Systems Unit Schematic

System Detail 2.2

Contents

Introduction

Details

General
Processor
Memory
Interrupt Control
Display Control
Floppy Disk Control
Expansion
Keyboard/Mouse data
System Reset
RS232 Communications
Parallel Printer Port
System Timer
Sound Generation
Port Addresses

Illustrations

1. System Board: data flow schematic

Introduction

The purpose of this chapter is to present an overview of the principal circuit elements on the System Board. The programmable elements are then broken down into further detail in subsequent chapters. This chapter also describes various miscellaneous areas of circuitry which do not warrant a chapter of their own.

The chapter also provides a full list of memory and I/O port addresses as allocated within the F1 system.

Details

General

The devices on the System Board, and the control, timing and interface circuitry associated with them, are interconnected with the Intel 8086 processor using the standard three parts of a 16-bit bus-based architecture; a 16-bit Data Bus; a 20-bit Address Bus; and a multi-purpose Control Bus.

Figure 1 represents this architecture in a block schematic diagram. It shows all the major peripheral areas of circuitry which are linked to the CPU. For simplicity, the diagram omits the detail of timing and control signals.

The processing system operated on the board is a real time interrupt driven system based on the interrupt structure of the 8086 and the interrupt facilities provided by two Zilog chips, a Z80 SIO and a Z80 CTC. There is no proprietary interrupt controller. The two Zilog devices operate together to provide all the necessary prioritised interrupt handling.

Other hardware functions and processes which require interrupt facilities are in turn "connected" to these two devices instead of being directly connected to the 8086 (apart from the disk controller which uses NMI).

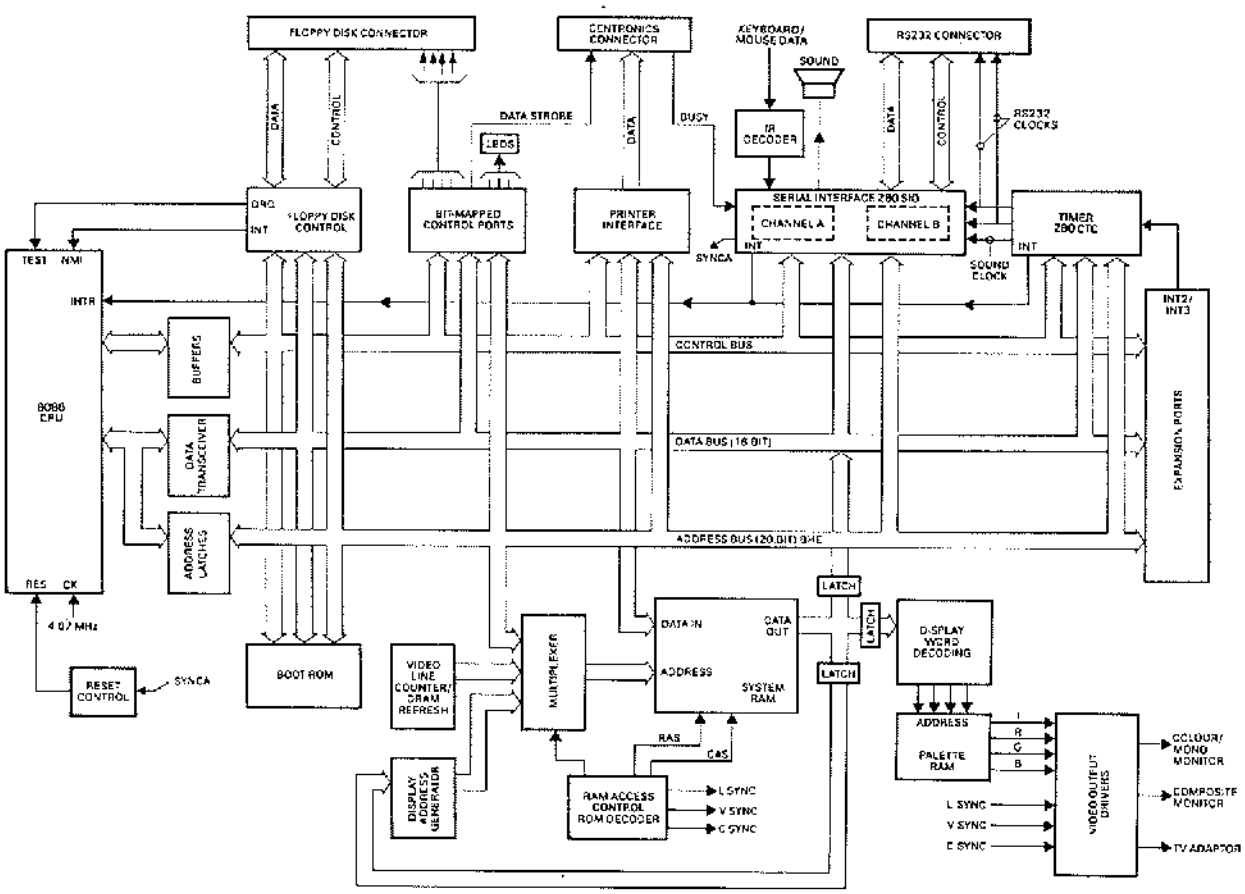


Figure 1. System Board: data flow schematic

Peripheral support for the 8086 is provided by a mixture of intelligent support chips and combinations of simpler standard logic elements.

These include:

1. A Western Digital WD2797-02 Floppy Disk Controller for controlling the MicroFloppy Disk Drive.
2. The Zilog Z80 SIO/2 which interfaces to the RS232C port, receives keyboard/mouse data, generates sound and also provides part of the interrupt structure.
3. The Zilog Z80 CTC which acts as a general system timer, determines the baud rates for the RS232C port and also provides part of the interrupt structure.
4. 256 Kbytes of DRAM (dynamic RAM — 512 Kbytes in the USA version). This is dual ported functioning as system RAM and bit-mapped Display RAM.
5. 32 Kbytes of ROM which store the code for the ROM based BIOS (expandable to 64 Kbytes).
6. A 16 x 4-bit static RAM which is integrated into the display control circuitry for selecting the colours/grey levels on the display monitor. This is termed the palette.
7. A parallel printer port constructed from discrete 74LS components.

Other areas of circuitry include; data latches, multiplexers, and transceivers; counters, timers, and decoders plus other pieces of "glue" logic. These devices are used in the supporting control circuitry for synchronising signals on the System Bus, generating DRAM and screen refresh addresses, and buffering transfers to external peripherals and devices connected on the Expansion Bus.

Processor

Details of the Intel 8086 processor are widely available. It is a true 16-bit processor, directly language compatible with the more commonly used 8088. It possesses:

1. 16-bit wide internal register architecture.
2. 16-bit wide external data bus.
3. Segmented addressing structure to support modular programming.
4. The capability of addressing 1 Mbyte of memory space and 64 Kbyte of system I/O.

The 8086 is configured to operate in minimum mode (i.e. it does not support a multi-processing configuration). It's basic clock frequency is 4.67 Mhz, which is derived from a 14 Mhz oscillator.

Wait States

A minimum of one wait state is automatically inserted into all memory and I/O transfer operations to and from the processor. This is extended in some cases to four wait states to suit the speed of slower peripherals (i.e. all peripherals mapped into the system I/O address range 00H to 3FH — see I/O address map at the end of the chapter).

CPU accesses to read from and write to locations in the System RAM (which includes the bit-mapped display RAM) are interleaved with accesses from two refresh circuits, which take precedence over processor accesses. One of the refresh circuits is used to refresh the Display; the second to refresh the DRAM array.

As processor accesses are asynchronous with the refresh cycles, the number of wait states inserted when accessing the Systen RAM is variable, being dependent on the refresh cycle (display refresh or DRAM refresh — if any) in progress. The minimum number of wait states inserted on a memory access is one; the maximum is 3.

External devices connected to the Expansion Bus can extend the basic one wait state to suit the slower speed of it's own peripherals, by activating the appropriate Expansion Bus input ready line (IORDY for I/O mapped peripherals; MRDY for memory mapped peripherals).

Memory

The F1 is fitted with a minimum of 256 Kbyte of system RAM (using 64K x 1-bit DRAMs). This is expandable by fitting one of the standard Apricot RAM expansion boards into the Expansion Slot. Initially in the USA only, the F1 is fitted with 512 Kbyte RAM as standard using 256K x 1-bit DRAMs. This is expandable to 768K using an Apricot 256K RAM Expansion Board.

In the 256 Kbyte model of the F1 range, the System RAM is formed by two banks each of 128 Kbyte; one mapped into the address range 00000H to 1FFFFH, the second into the address range 20000H to 3FFFFH.

Each bank is composed of sixteen 64K x 1-bit DRAMs (one bit per data line).

In the 512 Kbyte model of the F1 range, the System RAM is formed by a single bank of 256K x 1-bit DRAMs (half the board is left unpopulated). In this model The System RAM occupies the address range 00000H to 7FFFFH.

The system RAM is located at the bottom end of the available memory address range and is sub-divided into a number of allocated memory areas. These are specified by the hardware/BIOS as follows:

Address (hex)	Use
00000 — 003FF	Interrupt vectors
00400 — 01DFF	BIOS working area (pointers/ASCII image)
01E00 — 01FFF	Video Line Pointers
02000 — 0C7FF	Screen bit-image
0C800 — 0EFFF	BIOS data and stack
0F000 — 3FFFF *	DOS/application/user area

* 0F000 — 7FFFF on 512 Kbyte F1, both models expandable to BFFFFH.

Not all of the available System RAM is available to the user or application.

The lowest 1 Kbytes are reserved for the standard Intel interrupt vectors (4 bytes per entry).

42 Kbytes are reserved for mapping a bit-image of the screen (2 Kbytes are not normally used). 512 bytes are used to implement video line pointers. (Further details of display memory utilisation are provided in the section below on display control).

All other defined areas are allocated by the ROM based BIOS/operating system interface (see the "Guide to the BIOS" chapter for further details).

There is no proprietary DRAM controller for generating the necessary timing signals for accessing the DRAMs and refreshing the DRAM array. This is instead implemented by a series of 74LS counters, latches and other discrete logic elements. Refreshing the DRAMs is carried out during the display blanking period.

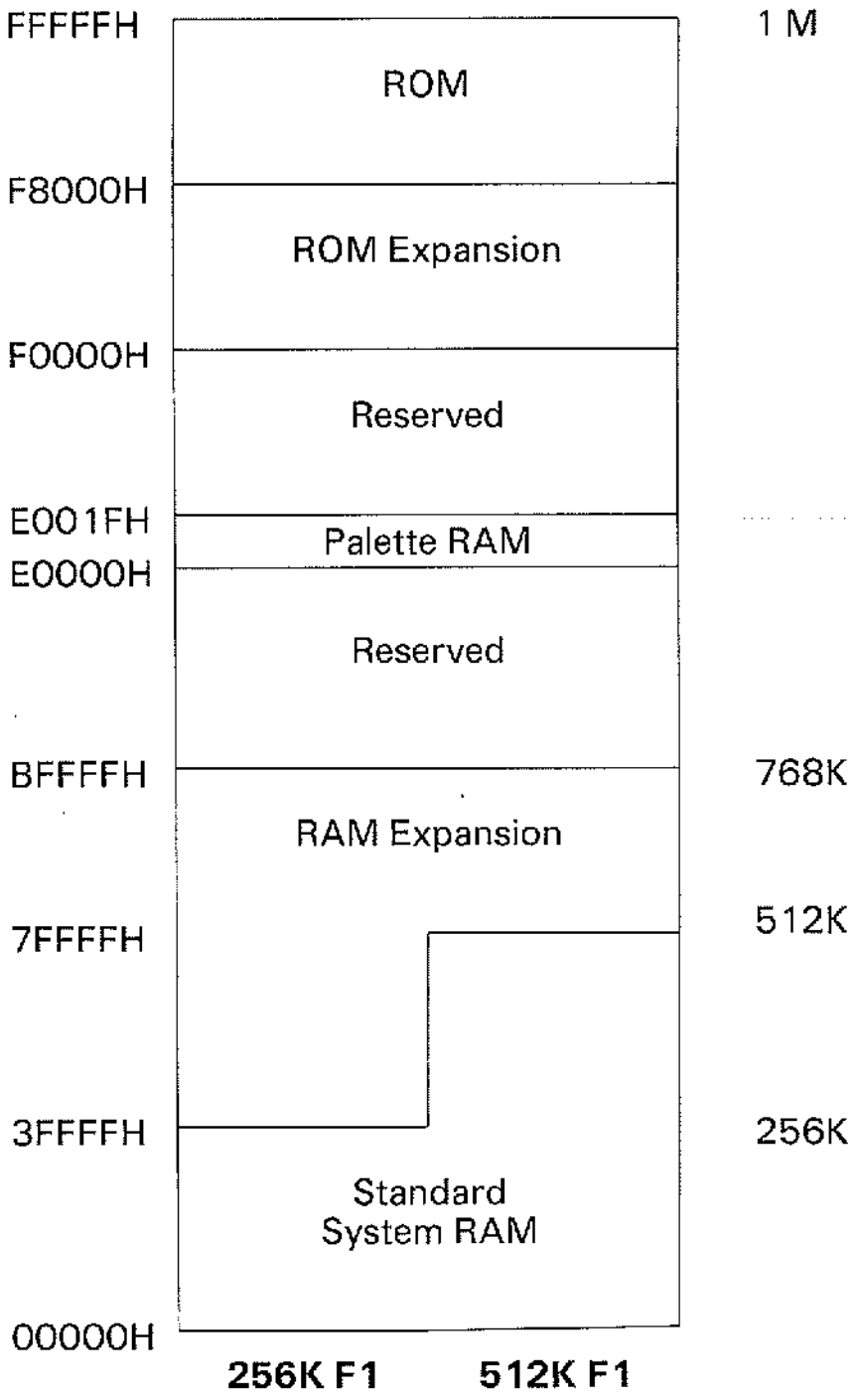
The System RAM area is not the only area of memory within the F1. Two other areas exist. One area is the palette RAM, the second is the Boot ROMs.

The palette is a small area of memory-mapped RAM formed by a 16 x 4-bit static RAM. It occupies 32 bytes of the available memory address space, starting at E0000H. (The RAM data lines are wired to the 4 LSB data lines of the system data bus). It is used to determine the colour mix at the display outputs. (This equates to grey level selection if a monochrome monitor is connected — see display control below).

The Boot ROMs are located at the top of the available memory addresss space. The Boot ROMs contain the bootstrap loader, diagnostics, calculator software and the BIOS code/ device driver routines for handling the standard hardware.

The BIOS code is currently contained in two 16K x 8-bit ROMs. The board is tracked to take 32K x 8-bit ROMs to allow for future BIOS expansion.

Memory Map


FFFFFH
1 M
ROM
F8000H
ROM Expansion
F0000H
Reserved
E001FH
Palette RAM
E0000H
Reserved
BFFFFH
768K
RAM Expansion
7FFFFH
512K
3FFFFH
256K
Standard
System RAM
00000H
256K F1
512K F1

Interrupt Control

The interrupt structure operated within the system is provided by the interrupt handling facilities within the Z80 SIO and the Z80 CTC. These two devices operate together to form a prioritised interrupt structure. Both devices can generate an interrupt and also produce an interrupt vector to indicate the cause of interrupt.

The Z80 SIO handles interrupt requests associated with:

1. Serial communications via the RS232C interface.
2. The keyboard/mouse data input channel
3. The sound output channel.
4. The Busy/Not Busy status control line from the printer port.

The Z80 CTC handles interrupt requests from devices on the Expansion Bus, and uses one of its own counter/timer channels to generate the System Clock interrupt.

These two controllers are connected together in a daisy chain configuration to prioritise interrupt requests to the Interrupt Request (INTR) input of the CPU.

A hardwired connection between the two controllers automatically sets the priority of interrupts from the SIO higher than those from the CTC. The interrupt output lines from the two devices are required together in wire-ORed fashion to form a single input to the CPU's INTR input.

The CPU responds to interrupt requests by implementing a two-stage interrupt acknowledge cycle, which causes the relevant controller to supply an interrupt vector onto the System Data Bus.

When the interrupt has been serviced, the programmer must implement a Return from Interrupt command sequence to clear the interrupt from the daisy chain.

The various interrupts can be enabled and disabled individually by program commands to the SIO and the CTC. All these interrupt sources can be masked as a group by clearing the Interrupt Enable Flag in the CPU. This action disables the Interrupt Request (INTR) input to the CPU.

The Non-Maskable Interrupt (NMI) input to the CPU has a higher priority than any of the maskable interrupts to the Interrupt Request (INTR) input. The NMI input is connected directly to the Interrupt Request (INTRQ) output of the Floppy Disk Controller (FDC). This output is activated whenever a disk read, write or formatting operation is successfully completed. It is also activated for certain errors in disk access operations.

Display control

The F1 has the ability to display information on a variety of different display monitors. These include:

1. An F1 colour monitor.
2. An F1 monochrome monitor.
3. A standard composite monitor.
4. A standard domestic colour TV (when fitted with the optional TV modulator).

The design of the display circuitry of the Apricot F1 is based on a bit-mapped architecture. There is no hardware differentiation between text and graphics; everything is pixel-based. i.e. A "dot" on the display screen is mapped by a corresponding bit(s) in the display memory.

The display memory is part of the system RAM and occupies 42.5 Kbytes in the lower 64K. 42 Kbytes are allocated in the system RAM to map out a pixel image of the display screen; the other 512 bytes are used to implement a series of 16-bit addresses which form a pointer to map each display scan line (video line pointers).

There is no high level CRT controller for generating display timing signals and display address lines. These are instead implemented by a variety of simple 74LS series components (counters, latches, etc).

The modes, resolutions and display features available to the programmer provided by the available display RAM are as detailed below. The resolutions described match the resolutions of the current ACT colour and monochrome monitors produced for the F1.

The F1 can be configured to drive either a colour or monochrome monitor with the programmer having the choice of displaying either 200 or 256 lines. The programmer also has one further option, either using an 80 column/640 pixel mode or a 40 column/320 pixel mode.

In the 640 pixel mode, the programmer can display up to 4 colours simultaneously (from a choice of 16) on a colour monitor, or up to 4 levels of greyscale if a monochrome monitor is connected instead.

In the 320 pixel mode, the programmer can display up to 16 colours simultaneously on a colour monitor, or up to a maximum of 8 levels of greyscale if a monochrome monitor is connected instead.

The 320 pixel/40 column mode is the mode which produces a sensible display output on a standard TV. (The relatively low bandwidth of a TV compared with a video monitor does not generally allow a sharply defined picture to be produced in the 640 pixel modes).

The table below summarises the display modes available on the F1.

Screen mode	Resolution	Max. No. of Colours Available
40 column	320 x 256	16 *
40 column	320 x 200	16 *
80 column	640 x 256	4
80 column	640 x 200	4

* This normally equates to 8 grey levels on a monochrome display due to the inherent limitation of the display in being able to successfully differentiate 16 grey levels.

The Display Control circuitry provides both the video drive signals and the necessary synchronising pulses which control the movement of the video beam across the screen to drive a variety of display monitors.

For colour monitors, the video signals are the digital IRGB outputs with horizontal and vertical scanning controlled by the signals HSync and VSync respectively. These are supplied via the 9-way D-type connector on the rear of the F1.

For the F1 monochrome monitor, the video signals are the digital RGB outputs (I is not decoded in the monitor due to the it's inability to produce 16 greylevels with acceptable differentiation). Horizontal and vertical scanning is controlled by HSync and VSync respectively. The signals are supplied via the 9-way D-type connector normally used for the colour monitor on the rear of the F1, together with + 17V (rectified a.c.) for powering the monitor.

The connection for a composite video monitor is the jack socket on the rear panel of the F1. This provides a composite video output consisting of a mixed video and sync signal, composed by combining the digital RGB signals with Hsync and Vsync, to produce an analogue greyscale equivalent. (As with the F1 monochrome monitor, the I signal is not used, with the result that only a maximum of 8 greylevels can be displayed).

The display circuitry cannot directly drive a standard domestic colour TV. This requires the optional TV modulator to be fitted into the internal Expansion Slot. When fitted, the TV modulator is supplied with the IRGB digital video signals and a combined horizontal/vertical sync signal via a Molex connector. These are then converted into the appropriate RF signal on board prior to being routed to the TV.

The Display Control circuitry consists of the following areas:

1. The screen RAM which holds the bit-image of the screen (part of the system RAM).
2. Pointer RAM (also part of the system RAM). This holds 16-bit addresses of the start of each line of video data stored in the screen RAM.
3. Various counters which are used to access the pointer RAM data and the screen RAM data, cycle through DRAM refresh addresses, generate horizontal and vertical sync pulses for setting the rate at which the video beam scans across the display, etc.
4. A pair of shift registers which decode the video data accessed from the screen RAM into a suitable code for driving the palette RAM.
5. The 16 x 4-bit palette RAM which translates the video data from the screen RAM into the appropriate colour/greyscale coded output.
6. The display output circuitry which converts the data accessed from the palette into the appropriate display signal(s) for driving the various display monitors.

Associated with the various counters is a State Timing ROM. This acts as a timing and control decoder. It is programmed to send out repeated sequences of timing signals, which control and coordinate the various accesses allowed to the System RAM (display refresh, CPU, DRAM array refresh).

Coded information about each pixel of the display is stored in the Screen RAM with each scan line of video data represented by a linear sequence of 80 words. The "80-word video lines" are accessed by the programmer writing the appropriate video line pointer data into the Pointer RAM. This equates to the address of the first word in each linear sequence of 80 words.

The number of bits used to represent each "pixel" on the screen is dependent on the mode selected. In 40 column mode, each "pixel" can be displayed in any one of 16 colours. This is represented by a 4-bit code in the system RAM. Each 4-bit code is translated into colour coded pixel data via the palette RAM. The palette RAM provides the programmer with the facility to translate the 4-bit code from screen RAM into whatever display pixel colour required.

In 80 column mode, each "pixel" can be displayed in any one of four colours from 16. This is represented by a 2-bit code in the system RAM. Each 2-bit code is translated into colour coded pixel data via the palette RAM. The programming of the palette handles the translation of the 2-bit code into the appropriate display pixel colour required.

The palette in the 80 column mode has to be programmed in a slightly different way than in the 40 column mode. As only two bits in the screen RAM are available to map each pixel in the 80 column mode, this code can only sucessfully represent one of four colours. The palette has 16 entries which can be programmed with any 4-bit value. The programmer restricts the output of the palette to one of four available colours by selecting four different values and programming blocks of four entries in the palette to the same value.

In the 40 column mode, no such restriction is replaced on programming the palette as the 4-bit code from the screen RAM can be used to represent any of one of the available 16 colours.

Sophisticated scrolling effects can be produced in the F1 by simply altering the video line pointer data. This includes left and right scrolling on character boundaries, up and down scrolling by a minimum increment of one video line.

Scrolling is implemented by simply updating the video line pointer data in the pointer RAM to access the appropriate area of screen RAM to be displayed and repainting any new image in Screen RAM as required. The video line pointers can also be used to produce fast screen blanking. This is simply achieved by modifying all the video line pointers to point to a blank line of data.

Floppy Disk Control

The Floppy Disk interface provides all the control functions necessary for formatting and transferring data to and from MicroFloppy Disks in the Disk Drive.

The configuration can operate with either single-sided or double-sided disks.

The Floppy Disk Interface consists of a Western Digital WD2797-02 Floppy Disk Controller (FDC), a series of buffers, a decoding circuit for selecting and engaging the disk drive heads, and the interface connector to the disk drive.

Four of the control lines to the Disk Drive are wired to a latch and are under direct control of the system software.

The remaining control and data transfer functions are implemented by the FDC. It controls the movement of the read/write head, transfers data to and from the disks, and monitors status signals from the drive.

All disk transfer operations performed by the FDC (formatting, reading disk data, writing data onto disks) are initiated by the 8086 CPU. The FDC then assumes control of the transfer operation to the disk.

Instead of employing a DMA controller to perform the data transfers between memory and the FDC, the F1 uses the CPU alone. The necessary data transfer speed is obtained by using a handshake mechanism utilising the TEST input pin on the CPU. This is described in the following paragraphs.

The FDC asserts DRQ (Data Request) to signify to the CPU that data is required on a write operation/data is available on a read. After initiating a transfer operation, the CPU polls this signal by monitoring its TEST input (using the WAIT instruction). When it detects an active DRQ signal, the CPU transfers the data byte (to the FDC when writing/from the FDC during a read). The CPU then restarts the polling of it's TEST input. The cycle then repeats for each of the bytes in the data transfer.

After the transfer of the last data byte, the FDC asserts INTRQ (Interrupt Request). This is wired to the NMI (Non-Maskable Interrupt) input line on the CPU and informs it that the transfer cycle has finished.

Expansion

The F1 has two expansion connectors, which enable it's basic processing system to be extended.

One connector is located internally within the F1 Systems Unit and has been designed to take a standard ACT Expansion Board. The second connector is accessible from the right hand side of the F1 Systems Unit and is designed to link in an external Expansion Unit.

A high degree of compatibility has been maintained with the other products within the Apricot range of computers. This is such that all existing ACT Expansion Boards (Winchester Controller, Modem, RAM cards, etc) can be used with the F1 without any modifications to the Expansion Board hardware.

The internal Expansion Slot and the external Expansion Connector are tracked onto the System Board and provide the following inputs and outputs:

1. The 16-bit System Data Bus.
2. The 20-bit System Address Bus.
3. Various control lines for interrupts and data transfers.
4. Power supply output(s).

The internal Expansion Slot is the same physical connector as used on other Apricot products (pc/xis and Portables). This is a 64-way connector (DIN 41612, 2 by 32 female, with a type B housing).

Of the 64 connections routed to the slot, 59 are compatible connections with the other products in the Apricot range mentioned above. There are minor differences in detail in these compatible connections. For example, the 15 Mhz clock output on the pc/xi corresponds to 14 MHz on the F1. The major differences are as follows:

1. There are no DMA facilities available on the F1 as provided on the pc/xi range of products.
2. The 8086 NMI line is not routed to the slot on the F1 since it is used within the system for disk transfers.

The external Expansion connector is a 60-way male IDC connector to which an external Expansion Unit can be connected. The connector is located on the right hand side panel of the Systems Unit, and mounted on the System Board.

The Expansion Unit is responsible for re-powering the Expansion Bus as necessary, to meet the drive capability of multiple Expansion Slots. Power supplies for the unit are not available on the connector apart from — 12V.

The connections wired to the Expansion connector are the same as the connections to the Expansion Slot apart from the supply lines + 12V and + 5V, which are not available.

Keyboard/Mouse Data

The receive channel of channel A of the Z80 SIO is used for keyboard/mouse data input. It is programmed to operate in synchronous mode (Monosync) at a data rate determined by the incoming data stream.

It is supplied with keyboard/mouse data via the IR receiver board which decodes the incoming data and converts it into an acceptable serial waveform. (Details of the IR Receiver Board are discussed in the chapter headed Systems Unit).

A signal conditioning circuit then separates the serial waveform into data and clock signals. The data is supplied to the receive data input of channel A (RxDA) and the clock signal to the receive clock input (RxCKA) to clock each data bit into the Z80 SIO.

The Z80 SIO converts the serial data into parallel format. It then generates an interrupt and produces an associated interrupt vector to signify keyboard/mouse data available.

The Keyboard formats a valid key closure into a serial packet consisting of a four byte sequence. The format is the synchronous transmission mode Monosync and is operated at a fixed data rate of approximately 3.85 Kbits/sec.

The four byte sequence consists of the Sync header byte (5AH), a status byte and two data bytes. The status byte and keycode data bytes are encoded with a Hamming format.

Using Monosync means that the Z80 SIO has to first detect a valid data pattern, (the sync header byte) before it regards the data sent as being valid. This provides a high degree of protection from other infra-red sources, as they will not contain the sync header and will therefore be totally disregarded.

Using a Hamming format to encode the data enables the BIOS software to check the integrity of the data received from the Keyboard. It produces a highly reliable system for proving the validity of the Keyboard data, providing a measure of protection against a transmission which contains a valid sync byte, but invalid data (missing or corrupted data).

The Mouse transmits data in a similar four byte sequence to the Keyboard. It uses the same sync header byte, but the data bytes provide information on mouse movement and the state of the mouse switches. The Z80 SIO is unable to differentiate between mouse and keyboard data. The BIOS software determines this by reading a flag bit in the synchronous data packet.

System reset

The system is reset by one of two methods; by powering the F1 off and on or by the System reset button on the Keyboard. Both methods produce the same effect on the circuitry. The reset control lines to the CPU and all peripheral circuitry are held active, setting all programmable elements to initialised status. When the reset line returns to it's inactive state, the CPU accesses the instruction stored at absolute address location FFFF0H to initiate the normal system start-up sequence.

The system reset button is located on the top edge of the Keyboard Unit. Holding the button down for approximately one second actions the reset circuitry on the System Board.

The actual "data" transmitted by the Keyboard to reset the system is a contiguous sequence of sync header bytes at a rate of one sync byte every 15.6 ms. These sync bytes are processed in the normal way of Keyboard data via the Infra-Red Detector Board, and clocked into Channel A of the Z80 SIO.

Each sync byte received by the Z80 SIO causes a pulse (logic low) on the Z80 SYNCA modem control output. In a normal data transmission, the sync byte is immediately followed by data and it's associated clocks which terminates the pulse on the SYNCA output.

When a string of sync bytes are sent with no data interleaved, the pulse on the SYNCA output is extended. (This effect is produced by feeding the SYNCA output back to the signal conditioning circuit which separates out the keyboard/mouse data from their associated clocks. Because no clock pulses are received following the sync byte, the SYNCA output alters the rate at which clock pulses are supplied to the SIO from the free-running oscillator in the conditioning circuit).

The extended SYNCA pulses gradually discharge a capacitor (C33) in the reset timing circuit. After a significant number of pulses, the capacitor is discharged sufficiently to trigger the output of the reset timing circuit (a 555 timer). This action generates the system reset signal.

RS232 Communications

The second channel (channel B) of the Z80 SIO is available for communication between the F1 and external equipment via an RS232C link.

The RS232C channel can be programmed to operate in either asynchronous or synchronous modes, with transmit and receive baud rates determined either via the Z80 CTC or via the external data communications equipment.

The RS232C interface is able to support:

1. Asynchronous communications with 5, 6, 7 or 8 bits per character and various choices of stop bits and parity sense.
2. Bit oriented synchronous communications, such as SDLC and HDLC.
3. Byte oriented synchronous communications, such as Monosync and Bisync.

The channel can also be programmed to generate an interrupt and also provide an associated interrupt vector for a variety of conditions occuring in the RS232 channel. These include:

1. Receive characters available.
2. External/status events (e.g. modem control line changes).
3. Transmit buffer empty.
4. Various error conditions (e.g. receiver overrun, parity errors, etc).

Parallel Printer Port

The Parallel Printer Port is designed to drive printers and plotters with a Centronics parallel interface.

The Centronics connector is wired for eight data output lines, and two of the handshake signals that are supplied on the majority of parallel printers; Data Strobe, and Busy/Not Busy.

The Printer Interface is designed using simple 74LS components and consists of:

1. An 8-bit latch, for the transfer of data bytes to the printer.
2. A control port for strobing data bytes into the printer (Data Strobe).
3. A printer status line (Busy/Not Busy) from the printer. This is wired to the Z80 SIO.

The Busy signal is wired to one of the input control lines of the Z80 SIO which is normally assigned as a Modem control input. Any transition on the Busy/Not Busy line signifying a change in the printer status, causes the SIO to generate an interrupt to the CPU and produce an interrupt vector.

System Timer

The System Clock interrupt is generated on a regular cycle of 20 ms by the Z80 Counter/Timer circuit (CTC).

The Z80 CTC is a multi-purpose timing device, with four programmable counter/timer channels and a prioritised interrupt structure. The channels are numbered from Channel 0 to Channel 3.

Channel 0 is programmed to respond to interrupt requests from the Expansion Bus. Channels 1 and 2 can be programmed to produce timing signals which are used by the Serial Input/Output (SIO) controller for RS232C communications and sound generation, respectively. Channel 3 is used by the BIOS for implementing the 20 ms system clock.

Sound Generation

The sound generator is a single channel "device". which can be programmed to generate simple audio tones, audio noise, or much more complex waveforms in the form of synthesised sounds, over the frequency range 600 Hz to 3 kHz.

The sound generator "device" is formed by two programmable elements which are also employed for a variety of other purposes as previously described. These are the two Zilog devices; the Z80 Serial Input/Output (SIO) controller, and the Z80 Counter/Timer Circuit (CTC).

The sound generator circuitry consists of a loudspeaker (16 Ohm, 0.4 W) driven by an audio amplifier. It is fed with a pulsed signal from the data transmit output (TxDA) of Channel A in the Z80 SIO.

The channel is programmed to operate in the byte oriented synchronous mode, Monosync. This matches the requirements of the data receive input of Channel A, which is used as the input for data from the infra-red keyboard.

The programmer can control the frequency, waveform shape, volume, and duration of audio output, and has the facility to produce either simple tones or complex synthesized sounds. The frequency of the audio output is set by the transmit baud rate for the channel, which is supplied from the System Board Timer (the Z80 CTC).

Port Addresses

Port Address	Function	Access
20H	SIO channel A data — keyboard/sound	r/w
22H	SIO channel A command — kbd/sound	w/o
22H	SIO channel A status — kbd/sound	r/o
24H	SIO channel B data — RS232	r/w
26H	SIO channel B command — RS232	w/o
26H	SIO channel B status — RS232	r/o
10H	CTC channel 0 — ext interrupt	r/w
12H	CTC channel 1 — RS232 baud rate	r/w
14H	CTC channel 2 — sound frequency	r/w
16H	CTC channel 3 — system timer	r/w
40H	FDC status register	r/o
40H	FDC command register	w/o
42H	FDC track register	r/w
44H	FDC sector register	r/w
46H	FDC data register	r/w
00H	Centronics data port	w/o
01H	Drive select	w/o
03H	Drive head load (active high)	w/o
05H	Drive motor on	w/o
07H	Video lines (1 = 200, 0 = 256)	w/o
09H	Video columns (1 = 80, 0 = 40)	w/o
0BH	LED 0 enable	w/o
0DH	LED 1 enable	w/o
0FH	Centronics strobe output	w/o
30H	Special dummy Z80 'RETI' port	w/o

Interrupt Control 2.3

Contents

Introduction

Details
- General
- Maskable Interrupt vectors
- Non-Maskable Interrupt (NMI)
- Programming the controllers
- Interrupt Control Sequence
- Maskable Interrupts

Programming

Illustrations

1. Interrupt Control

Introduction

There is no single, proprietary, interrupt controller in the F1. Instead, the system makes use of the interrupt handling facilities in two Z80 controllers: the Serial Input/Output Controller (SIO), and the Counter/Timer Circuit (CTC).

The Z80 SIO handles interrupt requests associated with:

1. Serial communications via the RS232C interface.
2. The keyboard input channel
3. The sound output channel.
4. The Busy/Not Busy status control line from the printer port.

The Z80 CTC handles interrupt requests from devices on the Expansion Bus, and uses one of its own counter/timer channels to generate the System Clock interrupt.

These two controllers are connected together in a daisy chain configuration to prioritise interrupt requests to the Interrupt Request (INTR) input of the CPU.

A hardwired connection between the two controllers (IEO to IEI) automatically sets the priority of interrupts from the SIO higher than those from the CTC. The interrupt output lines from the two devices are required together in wire-ORed fashion to form a single input to the CPU's INTR input.

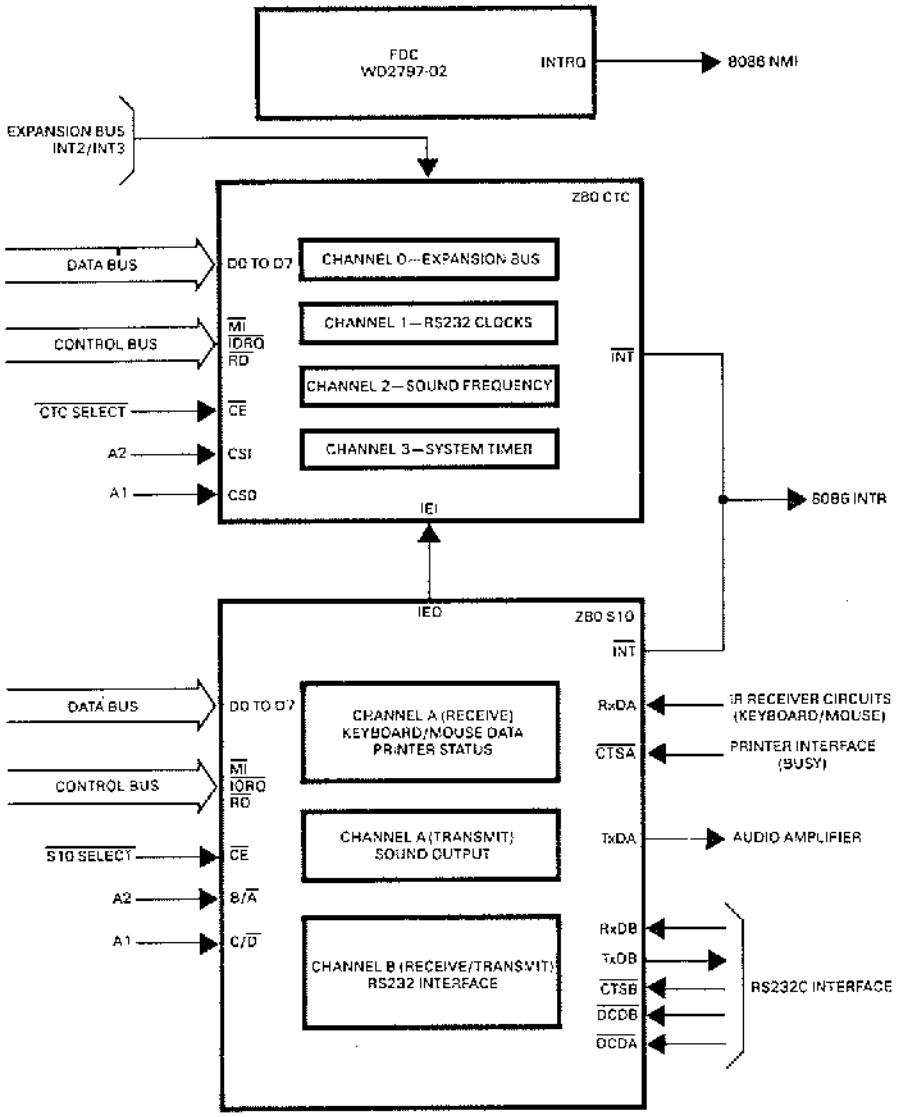


Figure 1. Interrupt Control

The CPU responds to interrupt requests by implementing a two-stage interrupt acknowledge cycle, which causes the relevant controller to supply an interrupt vector onto the System Data Bus.

When the interrupt has been serviced, the programmer has to send a Return from Interrupt command sequence to clear the interrupt from the daisy chain.

The various interrupts can be enabled and disabled individually by program commands to the SIO and the CTC. All these interrupt sources can be masked as a group by clearing the Interrupt Enable Flag in the CPU. This action disables the Interrupt Request (INTR) input to the CPU.

The Non-Maskable Interrupt (NMI) input to the CPU has a higher priority than any of the maskable interrupts to the Interrupt Request (INTR) input. The NMI input is connected directly to the Interrupt Request (INTRQ) output of the Floppy Disk Controller (FDC). This output is activated whenever a disk read, write or formatting operation is successfully completed. It is also activated for certain errors in disk access operations. Further details of this can be found in the Floppy Disk Interface chapter.

Details

General

Figure 1 shows the sources of the interrupt requests that can be programmed to generate vectored interrupts to the CPU.

The single Non-Maskable Interrupt (NMI) from the FDC to the CPU takes precedence over all other interrupts in the system. The CPU services this interrupt at the end of the current instruction, or between whole moves of a block-type instruction.

The sources which cause maskable, vectored interrupts to be generated by the Z80 controllers are listed below, in order of decreasing priority.

Interrupt requests vectored via the Z80 SIO

Keyboard input data received
Sound output buffer empty
Printer Busy/Not Busy transition

RS232C input data received
RS232C transmission buffer empty
RS232C modem control input signal transitions

Interrupt requests vectored via the Z80 CTC

Expansion Bus interrupt (INT2 or INT 3)
RS232C baud rate clock pulse *
Sound generation pulse *
System Clock interrupt

* These interrupts are not usually enabled

The interrupt output lines from the two Z80 controllers are connected together to form a single interrupt request line to the CPU.

The two controllers are also connected together directly, in order to control the relative priorities of interrupt requests along the daisy chain. The SIO takes higher priority than the CTC when queueing interrupt requests.

A line connects the Interrupt Enable Out (IEO) output of the SIO with the Interrupt Enable In (IEI) input of the CTC. When the SIO has cleared all its interrupt requests, IEO is set active (high) to enable the CTC to send interrupt requests to the CPU.

Within each controller, the relative priority of each type of interrupt is fixed as described below.

The Z80 SIO interrupts from Channel A take precedence over those on Channel B. This means that interrupts associated with keyboard input, sound output, and printer control, take precedence over those associated with the RS232C serial communications interface.

Within each of the two Z80 SIO channels, interrupt requests caused by data received take highest priority, followed by transmit interrupts, followed by external line/status events.

Within the CTC, Channel 0 has the highest priority, and Channel 3 the lowest. This means that interrupts from the Expansion Bus take precedence over the System Clock interrupt, which has the lowest priority in the system.

Maskable interrupt vectors

There are four possible sources of interrupts for each of the two channels in the SIO, and a further four possible sources in the CTC (one per channel).

To distinguish between the sources, a unique interrupt vector address is generated for each source within the appropriate Z80 controller.

Each controller has its own base vector address, which the BIOS writes to a register in the controller when the system is first initialised. When the controller generates an interrupt, it modifies the base vector address according to the cause of the interrupt.

The SIO base vector address is 50H. This value is written to Write Register 2 in Channel B of the SIO by the BIOS.

When an interrupt is generated in either Channel A or Channel B, this base vector is modified by increasing its value according to the cause of the interrupt, as shown in the table below.

Modified vector	Channel	Cause of Interrupt
50H	SIO B	Transmit buffer empty
52H	SIO B	External control/status event
54H	SIO B	Received data ready
56H	SIO B	Special receive status
58H	SIO A	Transmit buffer empty
5AH	SIO A	External control/status event
5CH	SIO A	Received data ready
5EH	SIO A	Special receive status

The CTC base vector address is 60H. This value is written to the Command Control Register in Channel 0 (see the Timer chapter for full details).

When an interrupt is generated in one of the four channels, the base vector is modified by the addition of 0, 2, 4, or 6 to its value, according to whether the interrupt occurred on Channel 0, 1, 2, or 3.

Non-Maskable Interrupt

At least one Non-Maskable Interrupt is generated by the FDC whenever it actions a disk transfer command. This means that the interrupts associated with disk transfer operations take priority over all other types of interrupt in the system.

The CPU services an NMI interrupt at the end of the current instruction, or between whole moves of a block type instruction. Another NMI interrupt is not accepted until the current interrupt has been acknowledged.

Programming the controllers

The Z80 controllers are initialised by writing data and control bytes to registers at ports in the System I/O Space.

The BIOS initialises the controllers at system load time. Command bytes to enable interrupts under specified conditions can be written to the controllers at any time subsequently.

Z80 SIO

The SIO has two sets of Write Registers, one set per channel, which control all its operations. Interrupt conditions are set by writing to these registers.

There is also a set of Read Registers in each channel. These record the current status of each channel following an interrupt.

Each channel of the SIO can be programmed to generate interrupts for any or all of the following conditions:

1. Either when it receives the first character of a block of data, or whenever it receives a character.
2. Whenever the data output buffer is empty during a transmission.
3. At any signal transition on the pins Clear To Send (CTS), Data Carrier Detect (DCD).
4. Various error conditions such as Transmit underrun, receive overrun, parity error, framing error.

Z80 CTC

The CTC has one Control Register and one Time Constant Register for each of its four counter/timer channels. Interrupts are enabled by writing to the Control Register. There is also a Downcounter Register in each channel, which records the state of the channel count when an interrupt within the channel occurs.

Each counter/timer channel of the CTC can be programmed to generate interrupts, in either counter mode or in timer mode. In both modes, the Downcounter Register is decremented to zero and then reloaded with a pre-programmed count value (the Time Constant) at a count rate determined by a clock source.

In timer mode, each decrement occurs on the leading edge of the CTC clock input (2.33 MHz). In counter mode, each decrement occurs on the falling edge of the clock/trigger input to that channel. If interrupts are enabled for that channel, an interrupt is generated internally whenever the downcount reaches zero.

Channel 0 is operated in counter mode. Interrupt signals from the Expansion Bus (INT2 and INT3) trigger Channel 0 to decrement a Time Constant of one to zero, whereupon an interrupt is generated.

Channel 3 is operated in timer mode to generate the 20 ms System Clock interrupt.

Channels 1 and 2 are not normally used for generating interrupts since Channel 2 produces the baud rate clocks for the RS232 interface, and Channel 1 provides the fundamental frequency for the sound output.

Interrupt Control Sequence

An interrupt request from either one of the controllers tells the CPU that an interrupt service routine is being requested, but not which interrupt service routine is required. To find out, the CPU automatically performs an interrupt acknowledge sequence, which provides an interrupt vector from the appropriate Z80 controller.

Further information to find the actual cause of interrupt when the vector indicates more than one possible source, is obtained by reading registers within the appropriate controller.

In the case of the Floppy Disk Controller (FDC) an interrupt on the CPU's NMI input causes a type 2 interrupt. The associated service routine can then check the cause of the interrupt by reading the FDC Status Register.

The FDC uses its Interrupt Request (INTRQ) output to signal that a disk read, write or formatting operation has been successfully completed, or to warn that an error condition has occurred during a disk access.

Full details of the conditions that cause the FDC to request an interrupt are given in the chapter on the Floppy Disk Interface.

In the case of maskable Interrupt Requests from the two Z80 controllers, the conditions which generate interrupts and the interrupt sequence are as follows below.

Maskable Interrupts

The interrupt sequence described here applies to all the maskable Interrupt Requests that can be generated by the two Z80 controllers, the SIO and the CTC.

When an interrupt condition occurs, the Z80 controller pulls its IEO (Input Enable Out) output low, so as to inhibit lower priority interrupts along the daisy chain. The controller then pulls its INT output low to assert an interrupt request to the CPU's INTR (Interrupt Request) input.

The IEO output of the Z80 CTC is not connected to another device so it has no effect on the operation of the circuit. The IEO output of the Z80 SIO is connected to the IEI input of the Z80 CTC and thus inhibits the CTC until all Z80 SIO interrupts have been serviced.

The CPU acknowledges an interrupt request by issuing two Interrupt Acknowledge (INTA) pulses. The 8086 interrupt acknowledge sequence is translated into a Z80 style interrupt acknowledge sequence by a series of logic gates.

The first INTA pulse causes the Z80 input M1 (Machine Cycle) to be pulled low. This acknowledges the interrupt request, and causes both Z80 controllers to freeze all current interrupt states so that the daisy chain can stabilise.

The second INTA pulse causes the Z80 input IORQ (Input/Output Request) to be pulled low. This causes the Z80 controller with the highest priority interrupt that is currently queued along the daisy chain, to supply the interrupt vector onto the data bus.

The CPU accesses the interrupt vector, and uses it as an offset into the Interrupt Pointers Table in the bottom of System RAM. The address of the entry in the Table is obtained by multiplying the vector by four. Each entry in the Table is a double word vector address which points directly to the appropriate interrupt service routine, using the standard 8086 addressing format. A list of the interrupt vectors in order of decreasing priority is given in tablular format below.

SIO Channel A

Interrupt Vector	Type	Input Pin	Source
5EH	Special Rx	RxDA	Keyboard data input
5CH	Rx data	RxDA	Keyboard data input
58H	Tx buffer empty	TxDA	Sound output
5AH	External	CTSA	Printer BUSY/ NOT BUSY
5AH	External	DCDA	RS232C control input (DCD)

SIO Channel B

Interrupt Vector	Type	Input Pin	Source
56H	Special Rx	RxDB	RS232C data input
54H	Rx data	RxDB	RS232C data input
50H	Tx buffer empty	TxDB	RS232C data transmitted
52H	External	CTSB	RS232C control input (CTS)
52H	External	DCDB	RS232C control input (DSR)

CTC

Interrupt Vector	Type	Input Pin	Source
60H	Channel 0	TRG0	Expansion bus
62H	Channel 1	—	Baud rate *
64H	Channel 2	—	Tone period *
66H	Channel 3	—	System Clock timer

* These interrupts are not usually enabled

The service routine can determine the cause of the interrupt (if ambiguous) or find out more information on the interrupt by reading status registers in the relevant controller. For the SIO, this is Read Register 0 or Read Register 1.

In the CTC, the Downcounter Register records how many clock inputs have been counted since the last interrupt for a zero count event. This is of value to the system clock interrupt service routine to enable it to determine the exact time of the system clock interrupt.

The interrupt service routine needs to reset the relevant bits in the status registers, so that they are clear to record the next interrupt. For the SIO, this is done by writing a reset command to Write Register 0. For the CTC, a reset command written to the channel's Control Register resets the channel accordingly.

At the end of the interrupt service routine, a two-byte Return From Interrupt (RETI) command, has to be issued to the two Z80 devices to allow lower priority interrupts to be serviced. The two bytes in the command are EDH, followed by 4DH. EDH forces IEO high to enable all unacknowledged interrupts lower down the device chain. 4DH releases the interrupt which has just been serviced.

The RETI sequence is generated by writing the two command bytes in sequence to port 30H in the System I/O Space.

If a device or channel is currently being serviced by an interrupt routine, then only devices or channels with a higher priority along the daisy chain can interrupt it. Lower priority interrupts are stored to await servicing when all the higher priority routines are complete.

Programming

Vectoring data for the Z80 controllers is initialised by the BIOS at system load time. Subsequently, the methods of selecting and enabling interrupts are dependent on the device generating the interrupt.

The FDC does not need to be initialised for interrupts. It generates Non-Maskable Interrupt requests as a result of the disk transfer commands that are issued to it.

The operation of the SIO, port addresses, the method of enabling and disabling interrupts, etc are all described in the chapter on the Serial Interface. The operation of the CTC, port addresses, the method of enabling and disabling the various interrupts, etc are all described in the chapter on the Timer. Full details of programming the FDC are given in the chapter Floppy Disk Interface.

The philosophy of using Expansion Bus interrupts (via the CTC) is detailed in the chapter on the Expansion Slot.

Since the interrupt structure is used extensively by the BIOS, care is required not to disturb control settings that are shared between the various interrupts that use the same device channels. For example, the keyboard input and the printer Busy line both use Channel A of the SIO.

It is also recommended that control settings should be restored to their previous state after use by other program routines.

Display Control 2.4

Contents

Introduction

Details

General
Display Modes and Features
Display Architecture
Drive Signals
Circuitry
Display RAM
Palette RAM
Pointer RAM
Mode Selection
Refresh Control and Timing
Display Connectors

Illustrations

1. Display control schematic
2. 40 column bit-map
3. 80-column bit-map
4. Pointer RAM data
5. Display Word decoding — 40 column mode
6. Display Word decoding — 80 column mode
7. Display period timing

Introduction

The display architecture of the Apricot F1 is a slightly different system to what is usually found on other microcomputers. It is totally graphics oriented (i.e. based on a bit-map instead of characters) and can also be software configured for a number of different resolutions and modes.

The display circuitry can also drive a variety of different display devices. These include:

1. An F1 colour monitor.
2. An F1 monochrome monitor.
3. A standard composite monitor.
4. A standard domestic colour TV (when fitted with the optional TV modulator).

Details

General

The design of the display circuitry of the Apricot F1 is based on a bit-mapped architecture. There is no hardware differentiation between text and graphics; everything is pixel-based. i.e. A "dot" on the display screen is mapped by a corresponding bit(s) in the display memory.

In other words, it does not matter whether the F1 is displaying text or graphics, the display circuitry treats them both in an identical manner. This feature of the design makes it easier for the programmer to mix text and graphics as required.

The display memory is part of the system RAM and occupies 42.5 Kbytes in the lower 64K. 40 Kbytes are allocated in the system RAM to map out a pixel image of the display screen; 512 bytes are used to implement a series of 16-bit addresses which form a pointer to map each display scan line (video line pointers); 2 Kbytes are not normally used.

There is no high level CRT controller for generating display timing signals and display address lines. These are instead implemented by a variety of simple 74LS series components.

Display modes and features

The modes, resolutions and display features available to the programmer provided by the display RAM are detailed in the next few paragraphs. The resolutions described match the resolutions of the current ACT colour and monochrome monitors produced for the F1.

The F1 can be configured to drive either a colour or monochrome monitor with the programmer having the choice of displaying either 200 or 256 lines. The programmer also has one further option, either using an 80 column/640 pixel mode or a 40 column/320 pixel mode.

In the 640 pixel mode, the programmer can display up to 4 colours simultaneously (from a choice of 16) on a colour monitor, or up to 4 levels of greyscale if a monochrome monitor is connected instead.

In the 320 pixel mode, the programmer can display up to 16 colours simultaneously on a colour monitor, or up to a maximum of 8 levels of greyscale if a monochrome monitor is connected instead.

The 320 pixel/40 column mode is the mode which produces a sensible display output on a standard TV. (The relatively low bandwidth of a TV compared with a video monitor does not generally allow a sharply defined picture to be produced in the 640 pixel modes).

Colour/greyscale selection is provided by a palette. This is a small area of memory-mapped RAM which determines the colour mix/grey levels at the display outputs.

Scan Line Modes

The 200 line modes have been implemented primarily for USA usage and other countries using 60 Hz mains supply frequency. The two 200 line modes as described previously are:

1. 640 x 200 bit-mapped graphics using any 4 colours from 16 (or 4 grey levels).
2. 320 x 200 bit-mapped colour graphics using 16 colours (or 8 grey levels).

The higher resolution 256 line modes are for UK, European and other countries using 50 Hz mains supply frequency and are as follows:

1. 640 x 256 bit-mapped graphics using any four colours from 16 (or four grey levels).
2. 320 x 256 bit-mapped colour graphics using 16 colours (or 8 grey levels).

Fonts

A default font of 128 characters (based within an 8 x 8 pixel cell) is contained in the system ROM. This is designed to be used with the 200 line resolution modes.

Each character is mapped by eight contiguous bytes in the ROM. A second font of 256 characters (7 x 7 characters based within an 8 x 8 pixel cell) is loaded into the system RAM at boot-up. Support in the BIOS also allows other 8 x 8 user-defined fonts to be installed within the system RAM. These can be easily accessed by simply modifying a font pointer.

A second default font of 256 characters (7 x 9 characters based within an 8 x 10 pixel cell) is loaded into the system RAM at boot-up. This is designed to be used with the 256 line display modes. Each character is mapped by ten contiguous bytes in RAM.

Support in the BIOS also allows other 8 x 10 cell user-defined fonts to be installed within the system RAM. These characters can also be easily accessed by simply modifying a font pointer. The 8 x 10 based font is of a greater resolution than the 8 x 8 based font for the 200 line modes but is of an identical 256 character set. The basic difference is in the construction of the characters, with lower case letters generally having longer descenders.

To obtain a sensible and usable "text mode" on both the colour display and monochrome display for existing text based applications, the attribute support by the BIOS is only allowed in "monochrome" on the colour monitor (i.e. any two colours from the possible sixteen) and any two grey levels on a monochrome monitor.

All the standard character attributes are available to the programmer in these two modes. These are produced by direct bit manipulation of the character image in the display RAM. Both normal and reverse video characters are supported with any combination of the following attributes:

1. Underline.
2. Strikethrough.
3. Intensity (simulated by shadow printing).

BIOS support for character attributes are not provided in the multi-colour modes due to the inherent nature of the colour display itself. (The same applies to the modes with more than two grey levels on a monochrome monitor).

Since the only effect an attribute is used for is to differentiate a character(s) from other characters, any of the standard attributes can easily be represented by assigning attributes to a different colour in a multi-colour mode (corresponding to a different shade of grey on a monochrome monitor), instead of the standard "monochrome" method.

Pointers

Sophisticated scrolling effects can be produced in the F 1 by simply altering the video line pointer data. This includes left and right scrolling on character boundaries, and up and down scrolling by a minimum increment of one video line.

Scrolling is implemented by simply re-ordering the video line pointer data in the pointer RAM to access the appropriate bit image to be displayed and repainting any new image in the Display RAM as required. The video line pointers can also be used to produce fast screen blanking. This is simply achieved by modifying all the video line pointers to point to a blank line of data.

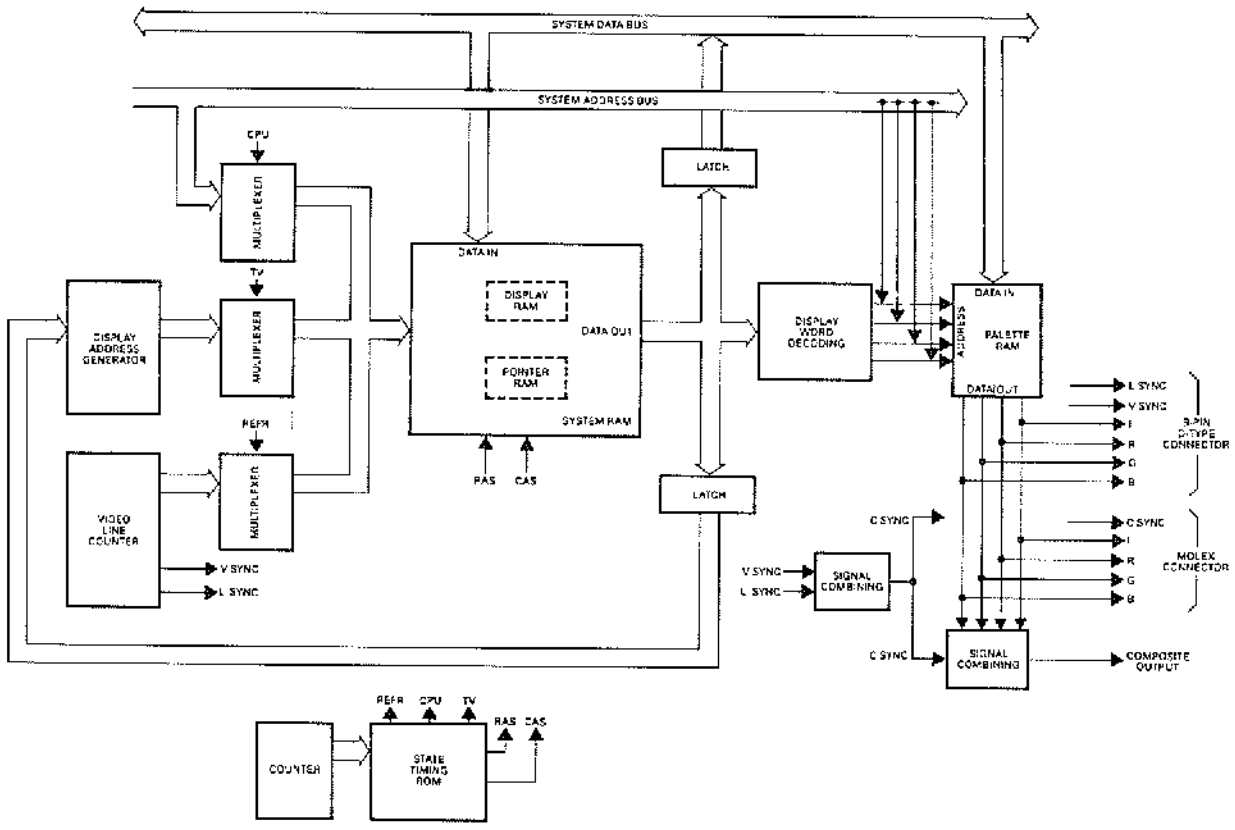


Figure 1. Display Control schematic

Drive Signals

The Display Control circuitry provides both the video drive signals and the necessary synchronising pulses which control the movement of the video beam across the screen, to drive a variety of display monitors.

For colour monitors, the video signals are the digital IRGB outputs with horizontal and vertical scanning controlled by Lsync (Line Sync) and VSync respectively. These are supplied via the 9-way D-type connector on the rear of the F1.

For the F1 monochrome monitor, the video signals are the digital RGB outputs (I is not decoded in the monitor due to it's inability to produce 16 greylevels with acceptable differentiation). Horizontal and vertical scanning is controlled by Lsync and VSync respectively. The signals are supplied via the same 9-way D-type connector as used for the colour monitor. The power for the monochrome monitor (+ 17V rectified a.c.) is also supplied via a pin on this connector.

The connection for a composite video monitor is the jack socket on the rear panel of the F1. This provides a composite video output consisting of a mixed video and sync signal, composed by combining the digital RGB signals with Lsync and Vsync, to produce an analogue greyscale equivalent. (As with the F1 monochrome monitor, the I signal is not used, with the result that only a maximum of 8 greylevels can be displayed).

The display circuitry cannot directly drive a standard domestic colour TV. This requires the optional TV modulator to be fitted into the internal Expansion Slot. When fitted, the TV modulator is supplied with the IRGB digital video signals and a combined horizontal/vertical sync signal (CSync) via a Molex connector. These are then converted into the appropriate RF signal on the Expansion board prior to being routed to the TV.

Circuitry

The Display Control circuitry consists of the following areas:

1. The Display RAM which holds the bit-image of the screen (part of the system RAM).
2. Pointer RAM (also part of the system RAM). This holds 16-bit addresses of the start of each line of video data stored in the Display RAM.
3. Various counters which are used to access the pointer RAM data and the Display RAM data, cycle through DRAM refresh addresses, generate horizontal and vertical sync pulses for setting the rate at which the video beam scans across the display, etc.
4. A pair of multiplexer/shift register stage which re-arranges the video data accessed from the Display RAM into a suitable code for driving the palette RAM.
5. The 16 x 4-bit palette RAM which translates the video data from the Display RAM into the appropriate colour/greyscale coded output.
6. The display output circuitry which converts the data accessed from the palette into the appropriate display signal(s) for driving the various display monitors.

Associated with the various counters is a State Timing ROM. This acts as a timing and control decoder. It is programmed to send out repeated sequences of timing signals, which control and coordinate the various accesses allowed to the System RAM (display refresh, CPU, DRAM array refresh).

Coded information about each pixel of the display is stored in the Display RAM with each scan line of video data represented by a linear sequence of 80 words. The "80-word video lines" are accessed by the programmer writing the appropriate video line pointer data into the Pointer RAM. The pointer equates to the address of the first word in each linear sequence of 80 words.

The number of bits used to represent each "pixel" on the screen is dependent on the mode selected. In 40 column mode (320 pixels per line), each "pixel" can be displayed in any one of 16 colours. This is represented by a 4-bit code in the system RAM. Each 4-bit code is translated into colour coded pixel data via the palette RAM. The palette RAM provides the programmer with the facility to translate the 4-bit code from Display RAM into any one of sixteen pixel colours as required.

In 80 column mode (640 pixels per line), each "pixel" can be displayed in any one of four colours from 16. This is represented by a 2-bit code in the system RAM. Each 2-bit code is translated into colour coded pixel data via the palette RAM. The programming of the palette handles the translation of the 2-bit code into the appropriate display pixel colour required.

The palette in the 80 column mode has to be programmed in a slightly different way than in the 40 column mode. As only two bits in the screen RAM are available to map each pixel in this mode, the code can only sucessfully represent one of four colours.

The palette has 16 entries which can be programmed with any 4-bit value. The programmer restricts the output of the palette to one of four available colours by selecting four different values and programming blocks of four entries in the palette to the same value (in effect masking out invalid entries in the palette).

In the 40 column mode, no such restriction is placed on programming the palette as the 4-bit code from the Display RAM can be used to represent any of one of the available 16 colours.

From a descriptive point of view, the display circuitry can be broken down into a number of different areas as detailed below. This division is purely a convenient way of describing how the circuitry is programmed. Each area of circuitry is expanded upon in further detail in the following pages.

1. Display RAM
2. Pointers
3. Palette RAM
4. Mode Selection

Display RAM

The display RAM is allocated to 42 Kbytes in the bottom 64K of System RAM. The start address is at 02000H, and stretches through to 0C7FFH. The maximum number of bytes to map a full screen of video data is 40 Kbytes (256 line mode).

Memory Planes

The display memory organisation for driving the various display outputs is based upon a system of memory planes. A plane is a general term for a block of memory in the display RAM. The number of planes in the system equates to the number of bits required to map each display pixel on the screen.

The planes in the F1 are not separated into contiguous areas in memory, but are interleaved at bit level within words.

Organisation of the memory into planes is entirely dependent on the mode selected. In the 640 pixel (80 column mode), the Display RAM is split into two planes.

In the 320 pixel (40 column mode), the Display RAM is split into four memory planes.

The display circuitry is programmed to regularly access a word of data from the Display RAM, as the raster scan on the display is in progress (i.e. the active display period). The display word accessed contains data from:

1. Both planes in the 640 pixel mode and represents 8 pixels.
2. All four planes in the 320 pixel mode and represents 4 pixels.

It is supplied via a multiplexer/shift register stage (formed by a pair of latches and two 4-bit shift registers) to the colour palette.

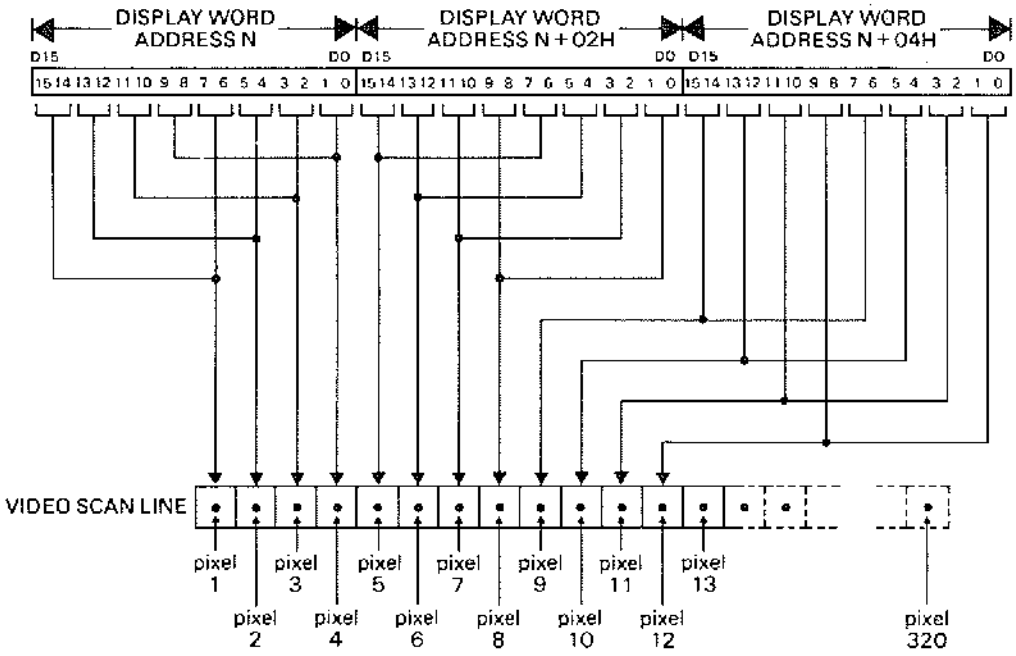
Usage of the planes is entirely dependent on how many colours or grey levels the programmer wishes to display on the monitor. (Note: The two terms colours and grey levels are generally interchangeable as it depends only on the type of monitor connected. This applies to all situations apart from the 16 colours in the 320 pixel mode. The circuitry within the display/on the board restricts the maximum number of grey levels to 8, but allows 16 colours).

16 colours requires 4 planes, 4 colours requires 2 planes, 2 colours (monochrome) requires a single plane. All other intermediate number of colours require the higher number of planes (e.g. 3 colours requires 2 planes). Masking out planes not in use is achieved by programming the palette accordingly.

40 column bit-map

In the 40 column mode, there is enough display RAM provided to allow the programmer to use 16 colours — i.e. four planes. Bits from each plane are extracted from each display word to form 4-bit code sequences to map each pixel. (Each video line consists of 320 pixels, with four bits mapped per pixel to produce the 16 possible different colour states. In the 256 line mode, the number of bits required to map the screen is therefore 320 x 4 x 256 bits = 40 Kbytes).

The four planes are aligned on alternate 2-bit boundaries in the display RAM. Each display word accessed, maps four pixels on the display screen. This is illustrated in Figure 2.



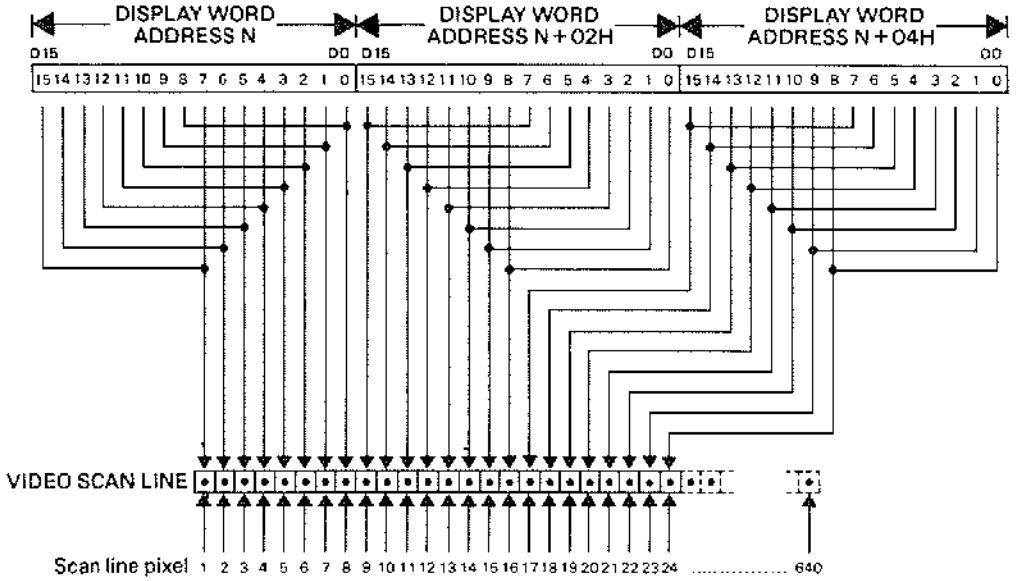
plane 1 bits	15	13	11	9
plane 2 bits	14	12	10	8
plane 3 bits	7	5	3	1
plane 4 bits	6	4	2	0

Figure 2. 40-column bit-map

80 column bit-map

In the 80 column mode, there is only enough display RAM provided to allow the programmer to use 4 colours — i.e. two planes. Bits from each plane are extracted from each display word to form 2-bit code sequences to map each pixel. (Each video line consists of 640 pixels, with two bits mapped per pixel to produce the 4 possible different colour states. In the 256 line mode, the number of bits required to map the screen is therefore 640 x 2 x 256 bits = 40 Kbytes).

The two planes are aligned on 8-bit boundaries in the display RAM. Each display word accessed, maps eight pixels on the display screen. This is illustrated in Figure 3.



plane 1 bits	15	14	13	12	11	10	9	8
plane 2 bits	7	6	5	4	3	2	1	0

Figure 3.80 column bit-map

Pointer RAM

The function of the Pointer RAM is to hold the start address of the first display word in each linear sequence of 80 words in display RAM, which are used to map each video scan line on the screen. The pointer RAM consists of 512 byte (256 words) in System RAM starting at address location 01E00H.

It can be regarded as a simple 256 entry pointer table, with one word per entry. All 256 entries are used in the 256 line scan mode. In the 200 line scan mode, only the first 200 entries are ever accessed.

The "80-word video lines" are accessed from the appropriate areas in the display RAM by filling the entries in Pointer RAM with the appropriate video line pointer data.

The first entry in the table points to the data to map the first video scan line. The second entry points to the data to map the second video scan line. The third entry points to the third line, the fourth entry to the fourth line, etc. The entries in the table are accessed by an address generated by a hardware controlled Video Line Counter.

At the end of each frame period, the Video Line Counter is automatically reset to point to the first entry in the pointer table. At the end of each active scan line (during the line blanking period), the Video Line Counter is automatically incremented to point to the next entry in the pointer RAM table.

During each frame period the following sequence of events takes place.

Prior to the start of each active line display period, the pointer data at the location specified by the Video Line Counter is accessed from the pointer RAM and is used to preset a second set of counters (termed the Display Address Generator).

The outputs of the Display Address Generator are used as the address for each display word stored in the display RAM. During the active display period everytime a display word is accessed, the Display Address Generator is automatically incremented to point to the next display word in the "80-word video line" sequence.

At the end of each scan line, during the line blanking period, the Video line counter is incremented. This new address is then used to access the next entry in the pointer table. The data accessed from the pointer table is then used to preset the Display Address Generator to point to the start address of the first display word in the next 80-word video line sequence.

This cycle continues throughout the whole of the active frame period. This equates to a 200 active line sequence in the 200 line mode; a 256 active line sequence in the 256 line mode.

Each entry in the pointer table locates the start of an 80-word video line sequence. Each "80-word video line" sequence may be contiguous areas of display RAM or in any other order as specified by the pointer data.

The 16-bit data in the pointer RAM does not equate to the absolute address of the first display word in each "80-word video line" sequence. The translation from the Display Address Generator to the multiplexed address lines for the RAM in effect produces a shift of one significant bit (the pointers access a whole word always requiring address bit 0 — A0 to be set low). This translation is illustrated in Figure 4.

The use of pointers makes it extemely flexible and easy for the programmer to perform text support features such a scrolling, screen blanking, etc.

Since the pointers point to a display word which represents a character width in 80 column mode (8 pixels), functions such as left and right screen scrolling can easily be performed.

For example to scroll a basic 80 x 25 text image one character to the left, all the programmer has to do is increment all the pointers by one word value and update the new display word which appears at the end of each 80-word video line.

Up and down scrolling is simply achieved by shifting entries upwards or downwards through the table and repainting each "new" 80-word video line as required. The minimum allowed change for scrolling is one scan line.

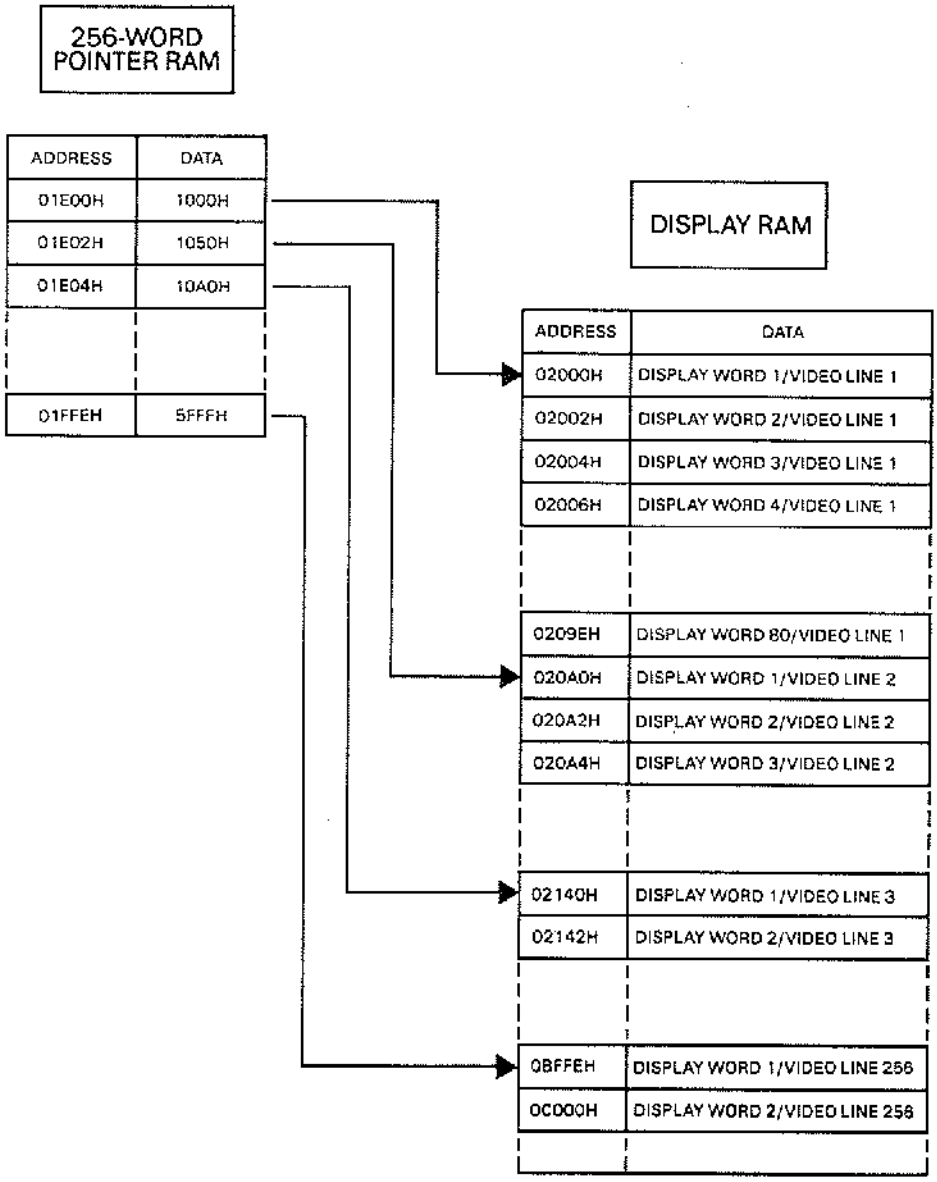


Figure 4. Pointer RAM data

Palette RAM

The palette is formed by a 16 x 4-bit static RAM, which is mapped into the system memory address space on word boundaries (even addresses) starting at E0000H. It can be regarded as a simple 16 entry table with 4-bits (LSB) active per entry.

The function of the palette is to translate the information from the planes into the colour coded IRGB outputs. It is dual-ported being continuously accessed by the data from the planes and is accessed by the programmer for setting up.

The colours produced by programming the palette with a particular 4-bit value are as detailed below. (Note: The palette data output lines are wired in the following order; the LSB data line — D0 to the Intensity output line; the next line — D1 to the Red output line; the next line — D2 to the Green output line; the MSB data line — D3 to the Blue output).

Palette data	Display Outputs B	G	R	I	Colour
00H	0	0	0	0	Black
01H	0	0	0	1	Dark gray
02H	0	0	1	0	Red
03H	0	0	1	1	Light red
04H	0	1	0	0	Green
05H	0	1	0	1	Light green
06H	0	1	1	0	Brown
07H	0	1	1	1	Yellow
08H	1	0	0	0	Blue
09H	1	0	0	1	Light blue
0AH	1	0	1	0	Magenta
0BH	1	0	1	1	Light magenta
0CH	1	1	0	0	Cyan
0DH	1	1	0	1	Light cyan
0EH	1	1	1	0	Light gray
0FH	1	1	1	1	White

40 column mode

In the 40 column mode, the programmer masks out the planes not in use and also selects the colours for display by programming the palette.

The translation of the data from the planes to colour-coded IRGB outputs is done by feeding the 4-bit parallel code sequence from the display RAM via a multiplexer/shift register stage to the four address inputs of the 16 entry palette RAM.

The function of the multiplexer/shift register stage is to sort the accessed display word into the correct 4-bit sequences to represent the display pixels (i.e. bits 15, 14, 7, 6 — first pixel; bits 13, 12, 5, 4 — second pixel; etc, see Figure 5).

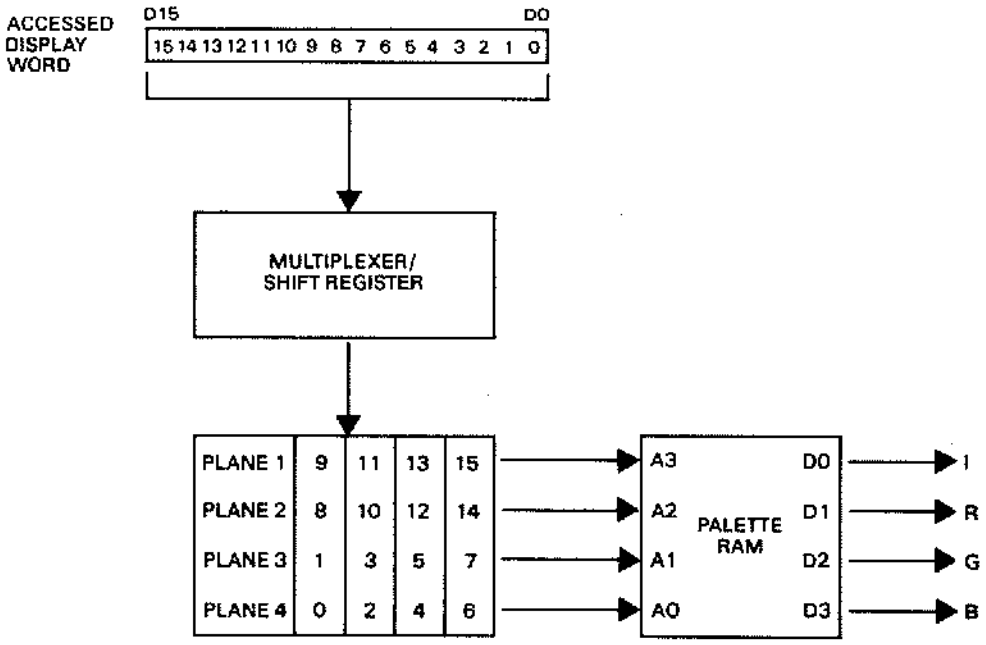


Figure 5. Display word decoding — 40 column mode

Data from plane 1 is supplied as the MSB address bit (bit 3) of the palette RAM, plane 2 data is supplied as address bit 2, plane 3 as address bit 1, and plane 4 as the LSB address bit (bit 0).

The 4-bit values programmed into the table set the colour selection at the output. If the programmer wants to mask out a plane (or planes), he does this by programming some of the entries in the palette with the same value. The entries which are programmed for masking are entirely dependent on the planes used for the colour display data.

This process of programming the palette is best illustrated by providing a few examples.

In order to display the full 16 colours on a colour monitor, all sixteen entries within the palette have to be programmed with different data, and all four planes filled with display data for driving the monitor.

The following table illustrates the required values for producing a full 16 colour display and the corresponding codes in the planes which address the entry in the palette.

Palette Address base offset *	Palette data BGRI	Plane Coding 1234	Colour
00	0000	0000	black
02	0001	0001	dark grey
04	0010	0010	red
06	0011	0011	light red
08	0100	0100	green
0A	0101	0101	light green
0C	0110	0110	brown
0E	0111	0111	yellow
10	1000	1000	blue
12	1001	1001	light blue
14	1010	1010	magenta
16	1011	1011	light magenta
18	1100	1100	cyan
1A	1101	1101	light cyan
1C	1110	1110	light grey
1E	1111	1111	white

* Address = E0000H + offset

The programmer does not have to program the palette to this colour selection, but can map them in reverse sequence or in any other permutation as required.

The main advantage of the mapping shown in the table above is that the translation from display data written into the 4 planes to the codes on the IRGB outputs, is a straightforward one to one relationship. Plane 1 data directly affects the state on the blue output line; plane 2 data affects the state on the green output line, etc. The 4-bit code accessed from the four planes therefore directly correspond to the codes on the IRGB outputs.

For an 8 colour display using planes 1, 3, and 4 only and masking out plane 2, the programmer has to program the palette using eight different values only, repeating each value twice in certain positions within the palette table.

An example to illustrate the entries that have to be programmed with the same value to mask out plane 2 is detailed below. The actual colour values chosen are an arbitrary 8 from the 16 available.

Palette Address base offset	Palette data BGRI	Plane Coding 1234	Colour Output
00	1111	0000	white
02	0001	0001	dark grey
04	1100	0010	cyan
06	0011	0011	light red
08	1111	0100	white
0A	0001	0101	dark grey
0C	1100	0110	cyan
0E	0011	0111	light red
10	1000	1000	blue
12	1001	1001	light blue
14	1010	1010	magenta
16	0000	1011	black
18	1000	1100	blue
1A	1001	1101	light blue
1C	1010	1110	magenta
1E	0000	1111	black

If you examine any two values which are programmed to the same colour value in the table above, you will see that the plane coding entries are the same apart from the third bit. This is the address line (A2) which is driven by the bit stream from plane 2.

The effect produced by placing the same colour value in this sequence is that any change of state made to data in plane 2 will be totally ignored. It is only the other planes which will produce a colour change as the bits from the planes change state. Changes to bits within plane 2 therefore will always be in effect ignored, always producing the same colour output irrespective of the state of the data stored.

To produce a monochrome display on the colour monitor, the palette has to be programmed with eight values specifying one colour and eight values specifying the other colour. The order which the values appear in the table depends on the plane assigned to store the display data.

An example of programming the palette using plane 1 for a monochrome output is illustrated below.

Palette Address base offset	Palette data BGRI	Plane Coding 1234	Colour Output
00	1111	0000	white
02	1111	0001	white
04	1111	0010	white
06	1111	0011	white
08	1111	0100	white
0A	1111	0101	white
0C	1111	0110	white
0E	1111	0111	white
10	1000	1000	blue
12	1000	1001	blue
14	1000	1010	blue
16	1000	1011	blue
18	1000	1100	blue
1A	1000	1101	blue
1C	1000	1110	blue
1E	1000	1111	blue

This will produce a white pixel everytime the bit is not set in plane 1 and a blue pixel when set (i.e. equating to a blue on white display).

To utilise plane 4 as the monochrome display every second entry in the palette is programmed with the same colour value; e.g. 00, 04, 08, etc for one colour; palette entries 02, 06, 0A, etc for the second colour. Both plane 2 and plane 3 can be used in the same manner, varying the programming of the palette accordingly.

Any combination of the 2 colours from 16 can be assigned to the colour monitor, by programming the palette with different values to produce a whole range of monochrome screens.

2-plane mode

In the 80-column mode, the programmer masks out the planes not in use, selects the colours for display and restricts the colour selection to four out of the possible sixteen by programming the palette.

The translation of the data from the 2 planes to colour-coded IRGB outputs is done by feeding the 2-bit parallel code sequence from the display RAM via the multiplexer/shift register stage to the address inputs of the 16 entry palette RAM.

The function of the multiplexer/shift register stage is to sort the accessed display word into the correct 2-bit sequences to represent the display pixels (i.e. bits 15, 7 — first pixel; bits 14, 6 — second pixel; bits 13, 5 — third pixel, etc, see Figure 6).

The action of the multiplexer/shift register stage is to always present a 4-bit code sequence to the address inputs of the palette. Only two bits are valid, the other bits are the code for the next "pixel" and are therefore invalid codes until shifted through the register.

Data from plane 1 is supplied as the MSB address bit (AB3) of the palette RAM, plane 2 data is supplied as address bit 1 (AB1). The entries in the palette have to be programmed so that whatever logic state appears on the other two address lines as data is shifted through the register, they do not affect the colour selection on the colour coded output lines.

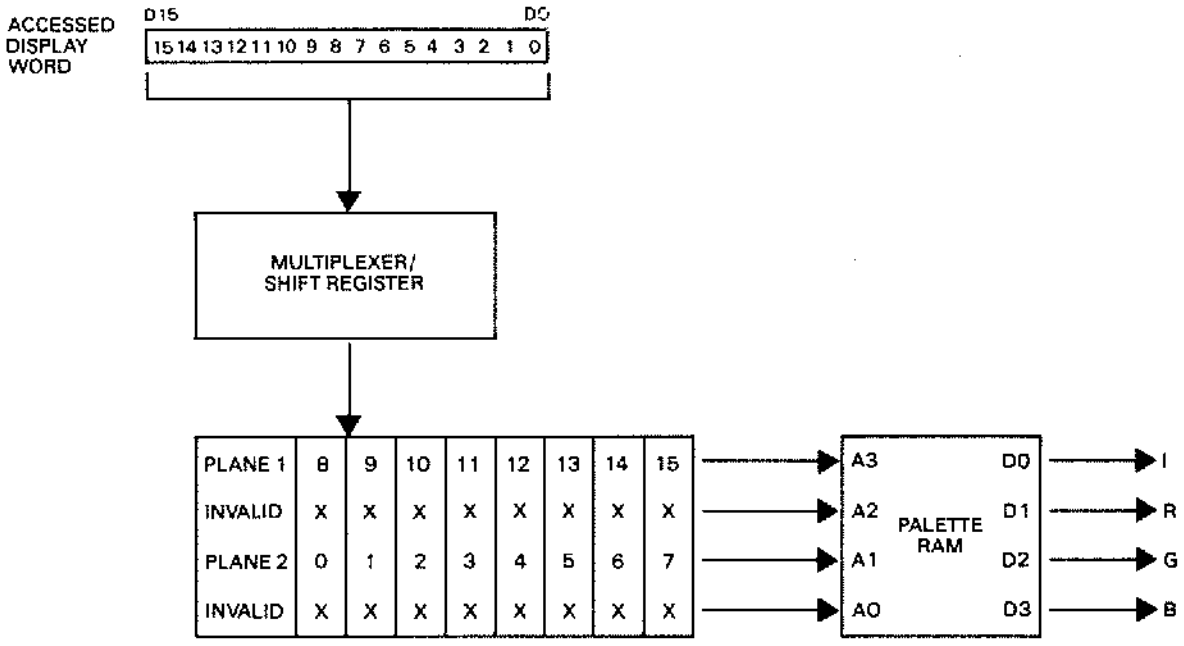


Figure 6. Display word decoding — 80 column mode

The programmer masks out these address lines by programming some of the entries in the palette with the same value. These entries are:

Plane Coding P1	P2	Palette Location Offset (hex)
0	0	00, 02, 08, 0A
0	1	04, 06, 0C, 0E
1	0	10, 12, 18, 1A
1	1	14, 16, 1C, 1E

This process of programming the palette is best illustrated by providing a few examples.

An example of programming the palette (and the corresponding codes required in the two planes to use four colours) is illustrated below. The actual colour values chosen are an arbitrary 4 from the 16 available.

Palette Address base offset	Palette data BGRI	Plane Coding P1	P2	Colour Output
00	0111	0	0	yellow
02	0111	0	0	yellow
04	0010	0	1	red
06	0010	0	1	red
08	0111	0	0	yellow
0A	0111	0	0	yellow
0C	0010	0	1	red
0E	0010	0	1	red
10	1000	1	0	blue
12	1000	1	0	blue
14	1111	1	1	white
16	1111	1	1	white
18	1000	1	0	blue
1A	1000	1	0	blue
1C	1111	1	1	white
1E	1111	1	1	white

An example of programming the palette using plane 1 for a monochrome output is illustrated below.

Palette Address base offset	Palette data BGRI	Plane Coding P1	P2	Colour Output
00	1111	0	0	white
02	1111	0	0	white
04	1111	0	1	white
06	1111	0	1	white
08	1111	0	0	white
0A	1111	0	0	white
0C	1111	0	1	white
0E	1111	0	1	white
10	1000	1	0	blue
12	1000	1	0	blue
14	1000	1	1	blue
16	1000	1	1	blue
18	1000	1	0	blue
1A	1000	1	0	blue
1C	1000	1	1	blue
1E	1000	1	1	blue

This will produce a white pixel everytime the bit is not set in plane 1 and a blue pixel when set (i.e. equating to a blue on white display).

To utilise plane 2 for the monochrome display every second pair of entries in the palette is programmed with the same colour value; e.g. 00, 02, 08, 0A, 10, 12, 18, 1A for one colour; palette entries 04, 06, 0C, 0E, etc for the second colour.

Any combination of the 2 colours from 16 can be assigned to the colour monitor, by programming the palette with different values to produce a whole range of monochrome screens.

Mode Selection

The control signals which set the display modes are provide by a pair of bit-wide control ports mapped in the System I/O space. They control the selection of:

1. 200 or 256 scan line mode.
2. 640 pixel (80 column) or 320 pixel (40 column) mode.

The scan line mode port is located at I/O address location 07H and is wired to the LSB data line on the high order section of the system data bus (D8). Writing 00H to the port selects the 200 scan line mode; writing FFH to the port selects the 256 scan line mode.

The column mode port is located at I/O address location 09H and is wired to the LSB data line on the high order section of the system data bus (D8). Writing 00H to the port selects the 40 column mode; writing FFH to the port selects the 80 column mode.

The output line from the scan line mode port is wired to the Video Line Counter circuitry. It's effect is to:

1. Determine the number of addressses generated by the Video Line Counter during the active line period (200 or 256).
2. Set the frequency of the vertical sync pulse (VSync) and vertical blanking waveforms to match the required scan mode (200 line — 60 Hz, 256 line — 50 Hz).

The output line from the column mode port is wired to a counter (via a NOR gate) in the video output driver stage. It's function is to control the rate at which changes occur on the IRGB colour coded outputs. In the 80 column mode (640 pixels per line), the maximum rate at which the lines can change state during the active display period (the pixel clock rate) is 14 MHz. In the 40 column mode (320 pixels per line), the rate is halved to 7 Mhz.

Refresh Control and Timing

Video control signals

The two output signals LSync and VSync, are generated to control the raster scanning across the screen display. They are synchronised with the various timing and control signals used for accessing the video data from the RAM.

LSync controls the horizontal scanning movement of the video beam, Vsync controls the vertical movement. Besides being supplied as separate signals to the 9-pin D-type connector, LSync and VSync are also combined to produce an interleaved line and frame sync waveform (CSync). This is supplied to the connector for the optional modulator and is mixed with a video greyscale signal to form a composite video waveform.

As the beam scans across the screen (during the active display period), the display outputs modulate it's behaviour and intensity so as to display pixels in the form as specified by the display data.

Two associated video timing signals are also produced on the board, a line blanking signal and a frame blanking (vertical blanking signal) — see Figure 7.

These are combined to produce a mixed blanking waveform. This is supplied to the counter in the video output stage. It's effect is to clear the display output lines (IRGB) to zero state during the blanking periods. It is also supplied as an address line to the state timing ROM for determining the allowed access cycles to the system RAM.

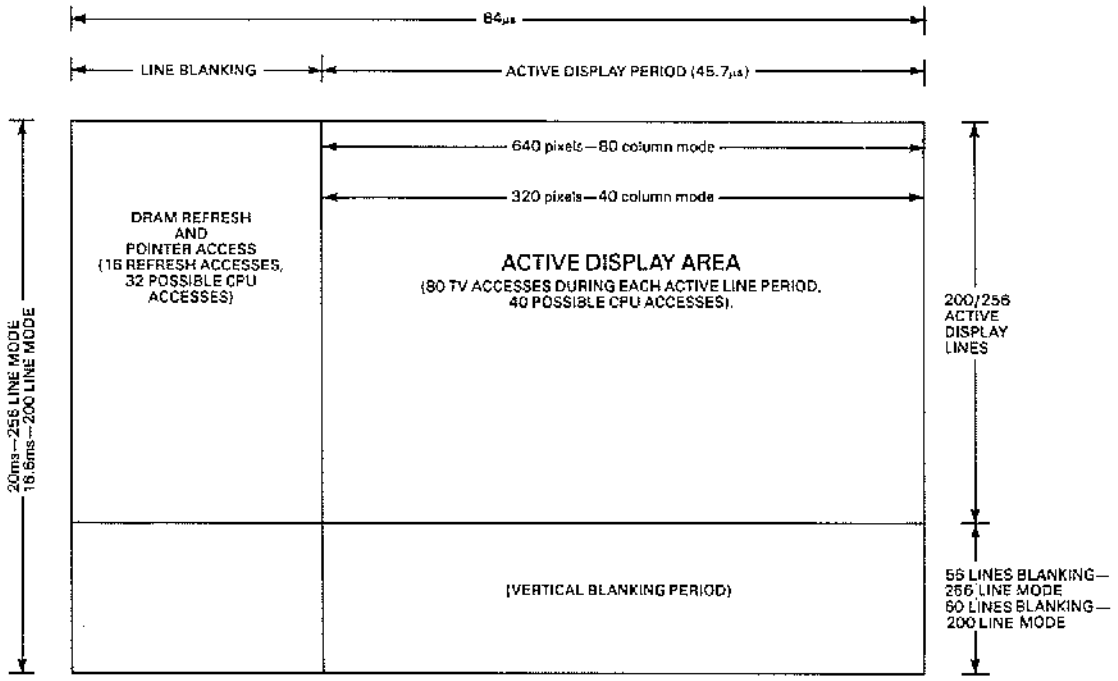

84μs
LINE BLANKING
ACTIVE DISPLAY PERIOD (45.7μs)
640 pixels—80 column mode
320 pixels—40 column mode
DRAM REFRESH AND POINTER ACCESS (16 REFRESH ACCESSES, 32 POSSIBLE CPU ACCESSES)
ACTIVE DISPLAY AREA
(80 TV ACCESSES DURING EACH ACTIVE LINE PERIOD, 40 POSSIBLE CPU ACCESSES).
(VERTICAL BLANKING PERIOD)
20ms—256 LINE MODE
16.6ms—200 LINE MODE
200/256 ACTIVE DISPLAY LINES
55 LINES BLANKING— 256 LINE MODE
60 LINES BLANKING— 200 LINE MODE

Figure 7. Display period timing

RAM Access Control

Various timing signals are produced on the board for co-ordinating and controlling accesses to the system RAM. These are synchronised to the video control signals and are derived from the basic on-board clock frequency of 14 MHz.

The timing signals are generated by a number of counters and a state timing ROM. This is nothing more than a 32 x 8-bit ROM programmed to act as a decoder. Its address inputs are derived from a 4-bit binary counter and the mixed display blanking waveform. It produces a series of pulse outputs throughout the active and blanking display periods to:

1. Generate the display refresh cycle.
2. Produce RAS and CAS pulses for latching addresses into the system RAM array.
3. Control all accesses to the RAM by generating the necessary control signals for enabling the various address multiplexers.
4. Determine when the CPU is allowed access to the system RAM.
5. Generate the DRAM refresh cycle.
6. Control the latching of data through to the palette RAM.

What process has access to the RAM is dependent on the period of the display cycle. During the active line periods, the display refresh cycle and the CPU are allowed access. During the blanking periods, the DRAM refresh cycle, the video line counter and the CPU are allowed access.

During the active line period, the display refresh cycle is guaranteed 80 accesses to enable 80 display words to be extracted from the display RAM for mapping each video scan line. CPU accesses have to be interleaved between the display accesses (TV accesses) and are only allowed when a display access is not in progress. A maximum of 40 CPU accesses are allowed in each active display period.

Any attempt by the CPU to access the RAM during a display word access causes a number of wait states to be automatically inserted to extend the CPU access time until the display access is complete. Wait states are inserted by the display circuitry driving the RAM Ready line low which is wired to the wait state control logic.

A minimum of one wait state is always automatically inserted on all CPU access (without driving the RAM Ready line). This can be extended to a maximum of 3 if a display cycle is in progress.

Each display word accessed takes eight 14 MHz clock cycles to clock the pixel codes through to the display outputs. The time taken to decode two Display Words is therefore sixteen 14MHz clock periods.

Each display word access from the RAM takes five 14 MHz clock cycles. Two display words take ten clock cycles. There is therefore time for three RAM accesses during the time it takes to decode two Display Words; two display word and one CPU.

This access sequence is determined by the State Timing ROM to give the two display word accesses, with a CPU access time slot (five clock pulses) allowed in between (i.e. TV CPU TV). Each of these "three-access" cycles includes an extra unallocated clock cycle, taking a total of 16 clock cycles in all.

The interleave sequence throughout the active line period is therefore; TV CPU TV, TV CPU TV, TV CPU TV, etc. During the period of each scan line there are 40 sequences of "three-access" cycles (80 guaranteed TV accesses, with 40 possible CPU accesses).

During the blanking periods, the DRAM array has to be refreshed and the Video Line Counter allowed one RAM access to enable the pointer data to be loaded into the counters which generate the addresses for each 80-word video line sequence.

A maximum of 32 CPU accesses can be interleaved between the DRAM refresh and video line counter access cycles during the line blanking period.

Any attempt by the CPU to access the RAM during a DRAM refresh cycle causes a number of wait states to be automatically inserted to extend the CPU access time until the DRAM referesh is complete. Wait states are inserted by the display circuitry driving the RAM Ready line low which is wired to the wait state control logic.

A minimum of one wait state is always automatically inserted on all CPU access (without driving the RAM Ready line). This can be extended to a maximum of 3 if a DRAM refresh cycle is in progress.

The DRAM array requires a RAS only refresh cycle to maintain the data stored within the RAMs. The refresh addresses are supplied via counters and a multiplexer.

Each line blanking period takes 256 clock cycles. 16 refresh cycles are allocated in this period so that the whole RAM array is refreshed within eight line blanking periods (i.e. 128 RAS refresh cycles). This takes approximately 0.5 ms, well within the required refresh limit.

The last DRAM refresh access in the blanking period (i.e. just before the start of the active line period) is used to access the pointer data from the RAM.

A single DRAM refresh cycle/video line counter access takes five clock cycles. The other clock cycles in each blanking period are allocated to CPU accesses, using the interleave sequence: CPU REFR CPU, CPU REFR CPU, etc.

This access sequence is determined by the State Timing ROM to allow two CPU accesses (five clock pulses each), with a DRAM refresh access time slot (five clock pulses) slotted in between (i.e. CPU, REFR, CPU). Each of these "three-access" cycles includes an extra unallocated clock cycle, taking a total of 16 clock cycles in all.

Display Connectors

F1 monitors

Both the F1 colour monitor and the F1 monochrome monitor connect to the F1 via the 9-pin D-type connector located on the rear of the Systems Unit. They are supplied with colour encoded video signals, and horizontal and vertical sync pulses.

The colour monitor is powered directly from it's own supply. The F1 monochrome monitor takes + 17V rectified a.c. from the System Unit via the 9-pin D-type connector. This is only present when the F1 monochrome monitor supply unit is plugged into the two-pin socket on the rear of the Systems Unit and the F1 is powered on. (The + 17V supply is switched through to the D-type connector via a relay controlled by the presence of the + 5V supply on the System Board).

The connections to the connector are detailed below.

Pin	Description
1	Not Connected for F1 Colour Monitor 17 Volts a.c. for F1 Monochrome Monitor
2	R
3	I
4	Horizontal Sync (LSync)
5	Vertical Sync (VSync)
6	Frame Ground
7	G
8	0V
9	B

Both Sync signals are at standard, positive TTL levels. The frequency of the Horizontal Sync pulses is 15.625 kHz; the frequency of the Vertical Sync pulses is either 50 Hz or 60 Hz, depending on the scan line mode selected.

All the video control signals, Intensity, Red, Green, and Blue are at standard TTL levels.

Composite monitor

A jack socket for a coax cable is provided at the rear of the System Unit for connecting a composite monitor.

The composite video signal is produced by combining the colour coded signals RGB with the combined sync signal (at the base of the transistor Q3). A resistive combining network is used to mix the RGB signals. The combined sync signal is produced by mixing horizontzl and vertical sync pulses.

The composite video signal output has a peak-to-peak voltage of 1.0 V. The sync pulses have a pulse amplitude of 0.17 V.

Domestic TV

A standard domestic television receiver can be connected to the F1 as a display screen but requires an optional modulator board to be fitted into the Expansion Slot. Connection to the TV aerial socket is then made via a socket on the modulator board and associated cable.

The input signals to the modulator are supplied from a 5-pin Molex connector (HD5), which is mounted on the System Board.

The connections are as detailed below.

Pin	Description
1	Composite Sync (CSync)
2	B
3	G
4	R
5	I

All the video control signals, IRGB and CSync are at standard TTL levels.

Expansion 2.5

Contents

Introduction

Details

General
Expansion Slot
Expansion Connector
Electrical Specification
Pin Detail
Address Allocation
Using Interrupts
Expansion Board Layout

Illustrations

1. Expansion Bus schematic
2. Expansion Slot
3. Expansion Connector
4. Expansion Board Detail

Introduction

The F1 has two expansion connectors, which enable it's basic processing system to be extended.

One connector is located internally within the F1 Systems Unit and has been designed to take a standard ACT Expansion Board. The second connector is accessible from the right hand side of the F1 Systems Unit and is designed to link in an external Expansion Unit.

To differentiate between these two connectors in the following description, the internal connector is referred to as the Expansion Slot. The externally accessible connector is referred to as the Expansion Connector.

A high degree of compatibility has been maintained with the other products within the Apricot range of computers. This is such that all existing ACT Expansion Boards (Winchester Controller, Modem, RAM cards, etc) can be used with the F1 without any modifications to the Expansion Board hardware.

Details

General

The internal Expansion Slot and the external Expansion Connector are tracked onto the System Board. The block diagram (Figure 1) shows how the System Bus and other control lines are wired to the Expansion connectors.
They consist of:

1. The 16-bit System Data Bus.
2. The 20-bit System Address Bus.
3. Various control lines for interrupts and data transfers.
4. Power supply output(s).

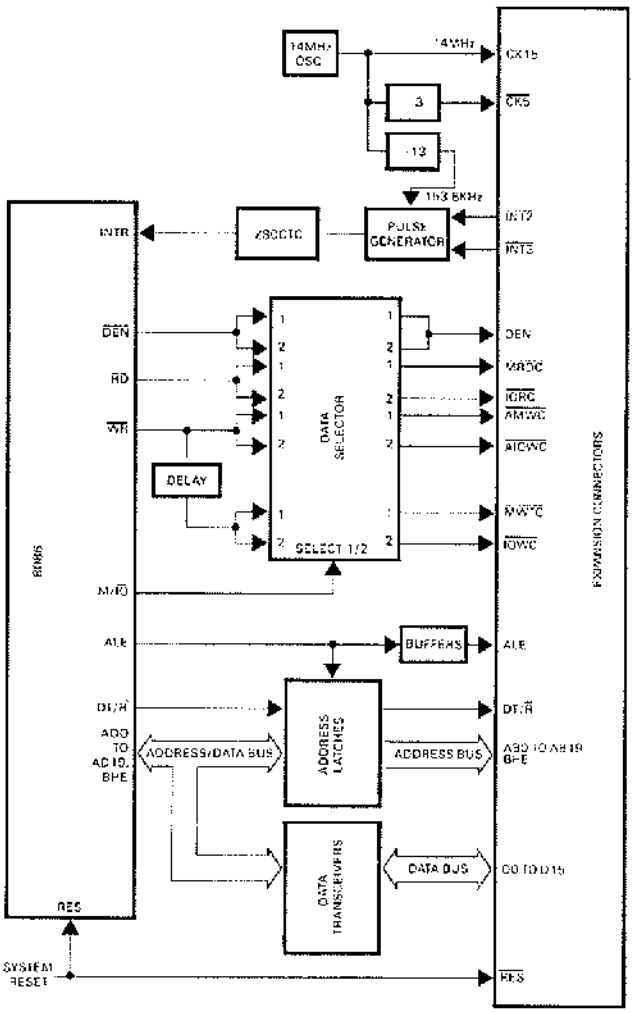


Figure 1. Expansion bus.

Expansion Slot

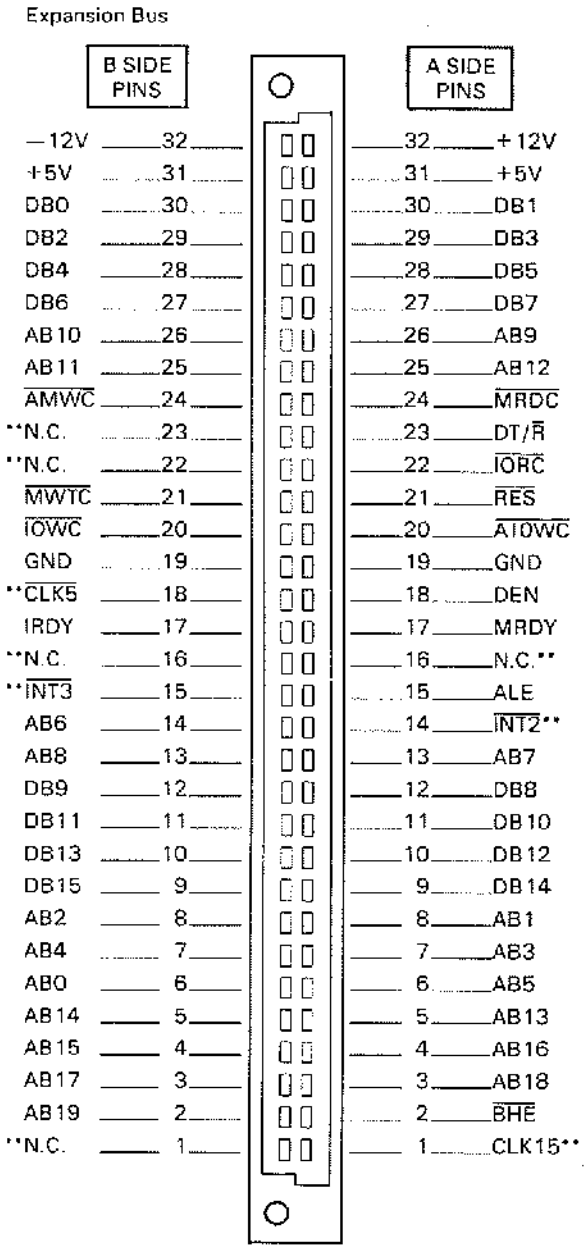
The internal Expansion Slot is the same physical connector as used on other Apricot products (pc/xis and Portables). This is a 64-way connector (DIN 41612, 2 by 32 female, with a type B housing).

Of the 64 connections routed to the slot, 59 are compatible connections with the other products in the Apricot range mentioned above. There are minor differences in detail in these compatible connections. For example, the 15 Mhz clock output on the pc/xi corresponds to 14 MHz on the F1 (see Figure 2).

The connections that are substantially different are also marked on Figure 2. The main area of differences are:

1. There are no DMA facilities available on the F1 as provided on the pc/xi range of products.
2. The 8086 NMI line is not routed to the slot on the F1 since it is used within the system for disk transfers.

An Expansion plate on the rear panel of the F1 can be removed to allow external connections to be made to Expansion Boards fitted into the Expansion Slot.



**Indicates differences with other products within the Apricot range. See Table below.

Pin	pc/xi	Portable	F1
A1	CK15	CK15	CK15 (14MHz)
A11	INT2	INT3	INT2
A16	EXT2	N.C.	N.C.
B1	NMI	NMI	N.C.
B15	INT3	INT5	INT3
B16	EXT1	N.C.	N.C.
B18	CK5	CK5	CK5 (4.67MHz)
B22	DMA1	N.C.	N.C.
B23	DMA2	N.C.	N.C.

Figure 2. Expansion Slot

Pin Definition - Expansion Slot/Connector

Pin	Description	Input/Output
AB0 to AB19	20-bit system address bus	Output
DB0 to DB15	16-bit system data bus	Bi-directional
$\overline{BHE}$	Bus High Enable	Output
ALE	Address Latch Enable	Output
$\overline{DEN}$	Data Enable	Output
DT/$\overline{R}$	Data Transmit/Receive	Output
$\overline{AMWC}$	Advanced Memory Write Command	Output
$\overline{MWTC}$	Memory Write Command	Output
$\overline{AIOWC}$	Advanced Input/Output Write Command	Output
$\overline{IOWC}$	Input/Output Write Command	Output
$\overline{MRDC}$	Memory Read Command	Output
$\overline{IORC}$	Input/Output Read Command	Output
MRDY	Memory Ready	Input
$\overline{IORDY}$	Input/Output Ready	Input
$\overline{RES}$	System Reset	Output
CLK15	14MHz Clock signal	Output
$\overline{CLK5}$	4.67MHz Clock signal	Output
$\overline{INT2}$	Interrupt Request 2	Input
$\overline{INT3}$	Interrupt Request 3	Input
+ 12V	System Board supply rail	Output *
— 12V	System Board supply rail	Output
+ 5V	System Board supply rail	Output *

* not available on the Expansion Connector

Expansion Connector

The external Expansion connector is a 60-way male IDC connector to which an external Expansion Unit can be connected. The connector is located on the right hand side panel of the Systems Unit, and mounted on the System Board.

The Expansion Unit is responsible for re-powering the Expansion Bus as necessary, to meet the drive capability of multiple Expansion Slots. Power supplies for the unit are not available on the connector apart from — 12V.

The connections wired to the Expansion connector are the same as the connections to the Expansion Slot apart from the supply lines + 12V and + 5V (see Figure 3).

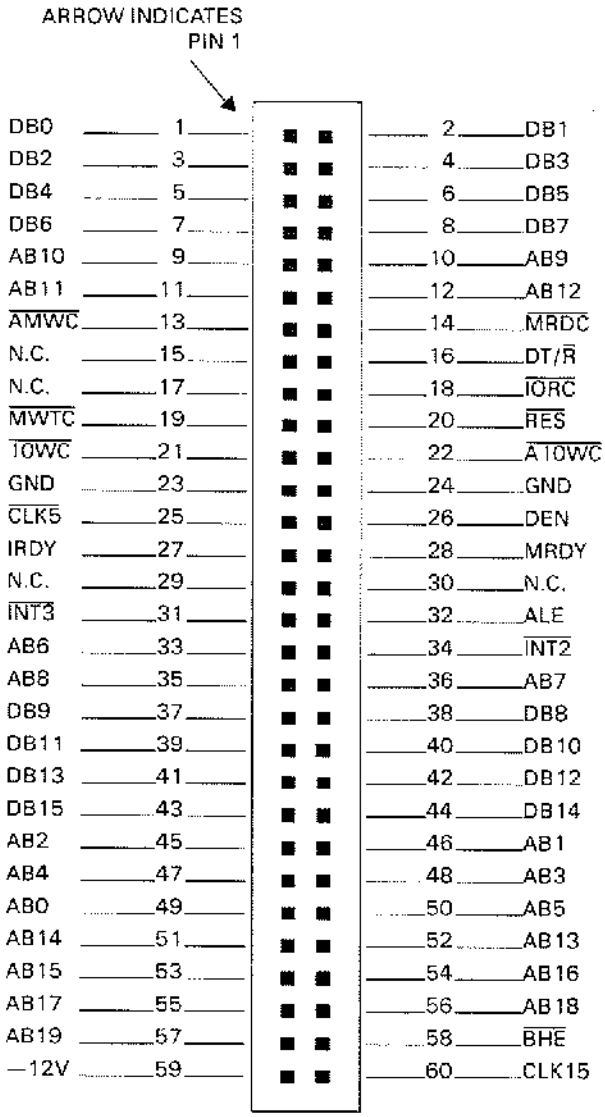


Figure 3. Expansion Connector

Electrical specification

The following paragraphs detail the standard electrical specification for an Expansion Slot within the current range of Apricot microcomputers. The same specification can be applied to the relevant connections on the F1 Expansion Connector.

Current Consumption

Maximum allowed current consumption of a circuit board fitted into the Expansion slot is:

0.5A from the + 5V rail.
50mA from the + 12V and — 12V rails.

Signal Outputs

All signal outputs (data, address, control and clocks) have the capability to drive a maximum of 2 LS TTL loads, i.e.

Logic high state voltage (Voh);

2.0 < Voh < 5.25 with maximum high state output source current of 40 μA.

Logic low state voltage (Vol);

— 0.5 < Vol < 0.8V with maximum low state output sink current of 0.8mA.

Signal Inputs

All signal and control inputs (apart from the interrupt lines) require a tri-state driver stage meeting the following requirements. Logic high state voltage (Voh);

2.4 < Voh < 5.25V with maximum high state output source current of 400 μA.

Logic low state voltage (Vol);

— 0.5 < Vol < 0.5V with maximum low output state sink current of 8mA.

The interrupt inputs require to be driven by an open collector driver stage. These lines on the System Board are fitted with pull-up resistors (3.3k).

Pin Detail

A description of each connection to the slot is detailed below.

DB0 to DB15	16-bit system data bus. Connected to the pair of transceivers which form the interface between the processor and the system data bus. DB0 is the LSB, DB15 the MSB. In effect, the bus is divided into two parts (the low order section DB0 to DB7, and the high order section DB8 to DB15), to support both 8-bit (byte) and 16- bit (word) data transfers. Byte transfers from/to even address locations are transferred on the bus lines DB0 to DB7 and byte transfers from/to odd address locations are transferred on bus lines DB8 to DB15. Word transfers from/to even addresses are transferred on bus lines DB0 to DB15 in a single operation. Word transfers from/to odd address locations are automatically transferred in two consecutive data byte transfer operations; the first operation uses bus lines DB8 to DB15 (odd address transfer), and the second operation use bus lines DB0 to DB7 (even address transfer).
AB0 to AB19	20-bit system address bus. Connected to the octal D-type latches which demultiplex the 20 address bits from the processor bus (time multiplexed address/data bus for the 16 LSB and the time multiplexed address/status bus for the 4MSB). AB0 is the LSB, AB19 the MSB. AB0 has a special function and is normally used in conjunction with the BHE signal to condition circuitry for byte or word data transfers.

Pin Detail continued

$\overline{BHE}$	Bus High Enable. Connected to an octal D-type latch which buffers the BHE signal from the processor. Normally used in conjunction with AB0 to enable circuitry for byte or word data transfers on the low or high order sections of the 16-bit data bus as follows; BHE AB0 Transfer operation 0 0 Whole Word 0 1 High order byte 1 0 Low order byte 1 1 None
ALE	Address Latch Enable. Driven by a pair of inverters. Negative edge of the active high pulse provides an indication of when the address is valid.
DEN	Data Enable. Output supplied by a line driver. Active high during memory and input/output data transfers.
DT/$\overline{R}$	Data Transmit/Receive. Processor output driven by an octal D-type latch. Signifies direction of data flow. Logic high indicates data transmission from the processing system; logic low, data reception.
$\overline{AMWC}$	Advanced Memory Write Command. Active low control signal which is set active before the Memory Write Command to provide memory-mapped devices an earlier indication of a write cycle.
$\overline{MWTC}$	Memory Write Command. Active low write command for memory-mapped devices.
$\overline{AIOWC}$	Advanced Input/Output Write Command. Active low control signal which is set active before the Input/Output Write Command to provide input/output devices an earlier indication of a write cycle.
$\overline{IOWC}$	Input/Output Write Command. Active low write command for input/output devices.
$\overline{MRDC}$	Memory Read Command. Active low read command for memory-mapped devices.

Pin Detail continued

$\overline{\text{IORC}}$	Input/Output Read Command. Active low read command for input/output devices.
MRDY	Memory Ready. Input connected to the wait state generator circuit on the Systems Board via an LS21 gate. Normally at logic high, but is set low to command the processor to extend the control transfer commands, by inserting wait states, until the selected memory- mapped device is ready for the data transfer operation. MRDY returning to logic high indicates that the selected memory-mapped device is ready for the data transfer operation (read or write).
IORDY	Input/Output Ready. Input connected to the wait state generator circuit on the Systems Board via an LS21 gate. Normally at logic high, but is set low to command the processor to extend the control transfer commands, by inserting wait states, until the selected input/output device is ready for the data transfer operation. IORDY returning to logic high indicates that the selected input/output device is ready for the data transfer operation (read or write).
$\overline{\text{RES}}$	System Reset. Output from the 555 timer circuit via an inverter. Active low state generated by the receiving a hardware reset (power on reset or via pressing the Reset button on the Keyboard).
CLK15	14 MHz Clock signal. Output from the 14 Mhz oscillator (50% duty cycle).
$\overline{\text{CLK5}}$	4.67 MHz Clock signal. Output from a divide-by-three circuit supplied by the 14 MHz oscillator. Inverted form of the processor clock signal. (66% duty cycle).

Pin Detail continued

$\overline{INT2}$	Interrupt Request 2. Input line merged with INT3 into a single interrupt line and wired to the clock/trigger input of channel 0 of the Z80 CTC on the Systems Board. The interrupt type number supplied to the 8086 processor on acknowledgement of an interrupt request on either INT2 or INT3 is 60H. (The interrupt type number acts as a pointer to the interrupt service routine).
$\overline{INT3}$	Interrupt Request 3. (see INT2 above).

Address Allocation

The available address locations in the system memory space and input/output space allocated to Expansion Boards are detailed in the following paragraphs. 8-bit devices connected to the lower half of the data bus must be located on even address boundaries and 8-bit devices connected to the upper half, on odd address boundaries.

System memory

The available address locations in the system memory space is dependent on the model within the Apricot F1 range.

40000H to C0000H - 256 Kbyte F1
80000H to C0000H - 512 Kbyte F1 *

* Initially only available in the USA.

1 processor wait state is automatically inserted on all memory read and writes. The processor wait states can be extended if the MRDY input to the system is utilised.

System input output space

The available I/O address range for the Expansion Boards is split into a number of categories according to the type of board. This approach has been adopted in-house within ACT for its whole range of Expansion products to avoid the possibility of the Expansion Slots being filled with Expansion Boards utilising the same I/O addresses.

Any third party vendor who wishes to design Expansion Cards which are/will be compatible with ACT's existing and future products should adhere to the same scheme.

The following table details the I/O addressses assigned to the various categories of Expansion Cards. (The values of the port addresses are in hexadecimal).

Expansion Board Category	I/O Port Address	I/O Port Address
Colour/Graphics Boards	80 to 8F	100 to 10F
IEEE 488 Controllers	90 to 9F	110 to 11F
Local Area Networks	A0 to AF	120 to 12F
Gateways	B0 to BF	130 to 13F
Miscellaneous Communications	C0 to CF	140 to 14F
ACT Reserved	D0 to DF	150 to 15F
Winchester Boards	E0 to EF	1E0 to 1FF
Modems	F0 to F7	1C0 to 1DF
Undefined	—	160 to 17F
Undefined	—	180 to 19F
Undefined	—	1A0 to 1BF

1 processor wait state is automatically inserted to any I/O read or write by accessing any of these addresses. The processor wait states can be extended if the IRDY input to the system is utilised.

Using Interrupts

Two interrupt lines are wired to the F1 Expansion Bus (INT2 and INT3). This is to maintain compatibility with other products in the Apricot range of computers, since the interrupt lines are merged into a single interrupt line on the System Board. The same interrupt type vector (60H) is supplied to the 8086 CPU irrespective of whether INT2 or INT3 generates the interrupt.

The two interrupt lines are connected together (via a pair of logic gates) to form a single physical interrupt line to the Z80 CTC on the System Board. The Z80 CTC is one of the devices capable of generating vectored interrupts within the F1. It is described in detail in the Timer and Interrupt Control chapters.

Reference should be made to these chapters if you require more low-level detail than provided in the paragraphs below. These describe how to integrate interrupt-driven Expansion Boards into the current range of Apricot computers.

To account for the possibility of more than one board requiring the use of the same Apricot interrupt line (and as there is only in effect one interrupt line on the F1 anyway):

1. All boards using either of the interrupt lines must contain a status register.
2. The interrupt must be driven by an open collector output.
3. The software device driver must operate in the manner described in the paragraphs below. The use of the status register will also become apparent.

If space permits on the board, the designer should track the board to both interrupt lines and fit a link facility to provide the board with the option of driving either interrupt line.

External Expansion Interrupt Set-up

To allow the software programer to easily link his device driver into all machines in the Apricot range of computers (Portables, F1 and pcs/xis with the generic BIOS), a special software interrupt is available. The purpose of this is to allow the programmer to mask out the differences in interrupt structure between the various machines within the Apricot product range with regard to interrupt type vectors.

The interrupt routine produced by generating an active interrupt on INT2 on the Apricot F1 is identified by interrupt type number 60H. The interrupt routine produced by generating an active interrupt on the corresponding interrupt line (INT3) on the Portable is identified by the interrupt type number 53H.

The Expansion set up interrupt 0F4H enables the programmer to install his vectors in the appropriate locations by relating them to the interrupt line without regard to the particular Apricot machine. (This applies to Apricot Portables, Apricot F1s and Apricot pcs and xis fitted with either the ROM BIOS or the RAM BIOS equivalent).

To use this interrupt requires the programmer to load the 8086 registers AL, BX and CX with the data as shown below and then generate the interrupt. Data is returned in BX and CX as shown. The purpose of this is explained in the next few paragraphs.

0F4H - External Expansion Interrupt Set-up

Input: AL = 0 - Set External Interrupt 2 (Winchester) *
AL = 1 - Set External Interrupt 3 (general) *
BX = Vector Offset Word
CX = Vector Segment Word

Output: BX = Old Vector Offset Word
CX = Old Vector Offset Segment

* See Figure 2 for corresponding Interrupt Lines on the Apricot Portable and the Apricot pc/xi range of machines. The interrupt line INT2 on the F1 corresponds to INT2 on the Apricot pc/xi range of machines. This was originally reserved for use solely by the Apricot Winchester Controller. This is no longer a requirement if machines are fitted with Revision 9 or later versions of Winchester Controller Board.

Installing the device driver at system boot

One of the routines executed during initialisation must be to install the two vector words associated with the device driver service routine using the Expansion Set-up interrupt. The old vector addresses returned must be saved (the reason for doing this is given below).

Run-time operation

The first task undertaken by the service routine must be to read the status register to check that the device is the actual cause of the interrupt. If not, the device driver should immediately relinquish control to the service routine specified by the vectors returned by the Expansion Set-up Interrupt.

If the interrupt is caused by its own device as indicated by the status register, the driver naturally performs the appropriate service routine. At the end of the routine, the device driver must relinquish control to the routine specified by the vectors returned by the Expansion Set-up Interrupt.

This effect, where multiple device drivers are installed all using the same interrupt line, is similar to the technique of "daisy chaining" interrupt lines and acts as a logical extension to the MS-DOS 2.0 installable device driver philosophy.

Notes

1. At boot-time the BIOS assigns in effect a null device driver to the available interrupt numbers, which performs two functions:

   a. Supplies the RETurn from Interrupt (RETI) command sequence to the Z80 Controllers.

   b. Returns control back to the program point of interruption.

The "null device driver" is always the last driver to be serviced, since it will always be the first one installed (i.e. the end of the "daisy chain").

2. Since the last loaded driver will always be the first device serviced following an interrupt (i.e. the beginning of the "daisy chain"), the order of loading multiple device drivers automatically assigns an order of priority. Time critical device drivers therefore, should always be loaded last.

3. The maximum time allowed for each interrupt service routine should generally not exceed 10ms.

Expansion Board Layout

The dimensions and layout details for an Expansion Board are detailed on Figure 4.

The uppermost view illustrates the overall board dimensions and the location of the connector.

The middle projection provides a different view of the connector and details the maximum height available for components mounted on the board.

The lower illustration details all the drilling requirements for the printed circuit board and the board area available.

This design of Expansion Board is for a standard board which will fit into the Expansion Slots on the dual expansion slot version of the ***Apricot pc/xi*** range of micros. These are currently the machines which are most restricted in terms of space available for Expansion components.

In order to design boards which encompass the full range of Apricot products, all boards should be designed to these tighter tolerances. (In general, all Expansion Boards should be designed to the tighter tolerances specified for the slot Expansion 1 on the ***Apricot pc/xi*** range to allow other boards which may require the extra depth of Expansion 2 slot to fit into the pc/xi machines).

Note:

1. The 3.85 mm diameter holes located in three corners of the board are only required on expansion cards made up of two boards, in the form of a "sandwich "(i.e. a main board fitted with the expansion connector, and a piggyback board separated from the main board by spacers). In this situation, the three holes can be used as general purpose tooling holes and also as screw holes for the spacer fixings.
2. On single board expansion cards, the 3.85 mm diameter located in the upper right-hand corner is not required, thus providing a small extra area of board space. The two 3.85 mm holes on the left-hand side of the board together with the connector fixing hole in the lower right-hand corner can then be used as tooling holes, if required.

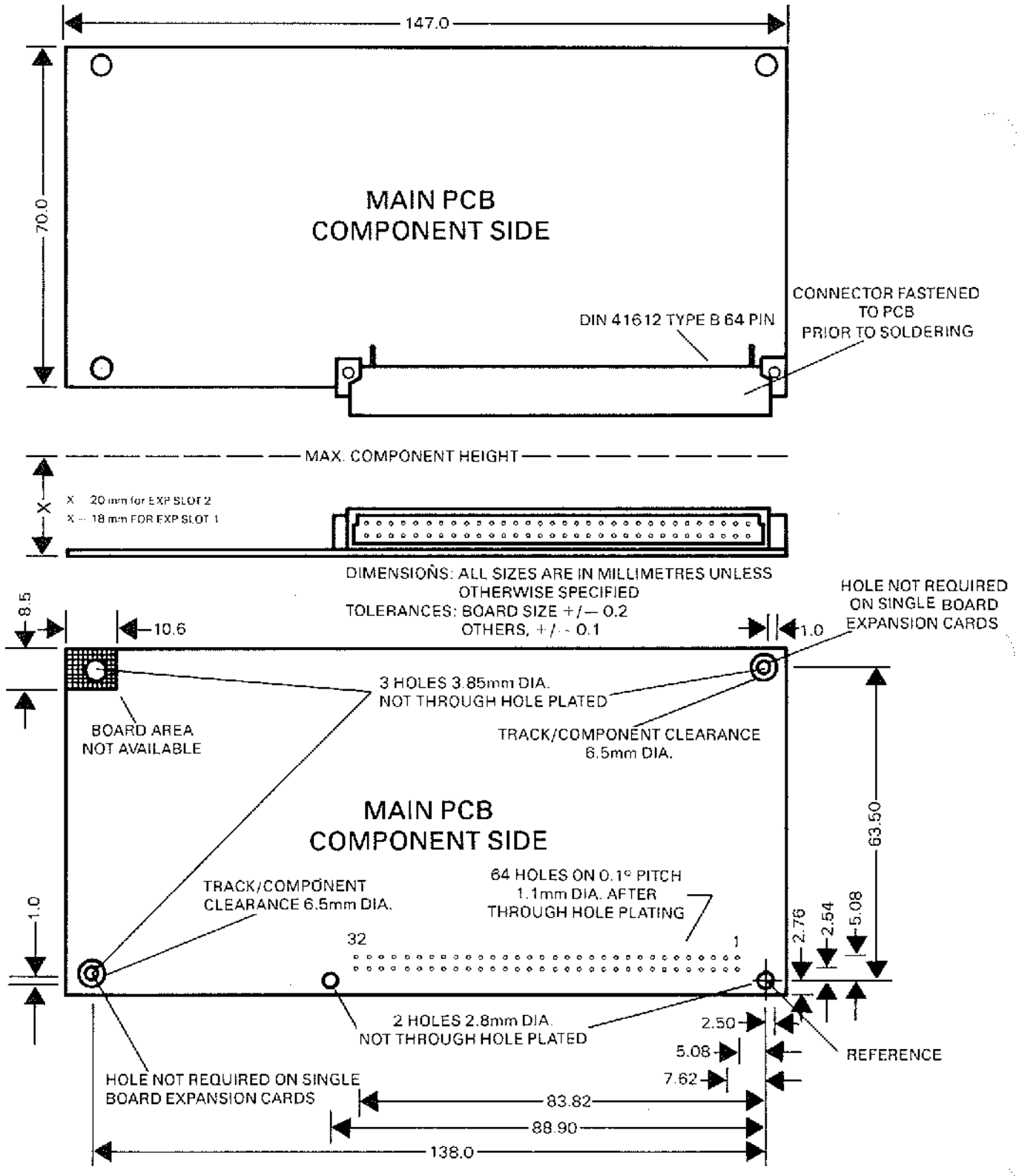


Figure 4. Expansion board detail for ***pc/xi*** range

Floppy Disk Interface 2.6

Contents

Introduction

Details
- General
- Disk Write
- Disk Read
- Disk formatting
- Read/write head positioning

FDC detail
- General
- Processor Interface
- Data Requests
- Interrupt Requests
- Disk Drive Control
- Command Register
- Status Register
- Track Register
- Data Register

Programming Considerations
- General
- Disk Drive Selection
- Motor Control
- Head Loading
- Head Positioning
- Data Transfers
- Formatting Commands
- Force Interrupt Command

Interface Connection Detail
- System Connections
- Disk Drive Connections

Track Format

Illustrations

1. Floppy Disk Interface
2. Track format

Introduction

The Floppy Disk Interface is located on the System Board and consists of the elements of circuitry as illustrated on the block diagram, Figure 1. The interface provides all the control functions necessary for formatting and transferring data to and from MicroFloppy Disks in the Disk Drive.

The configuration can operate with either single-sided or double-sided disks.

The disks are soft-sectored and encoded with the same disk format as the disks used on the Apricot pc/xi range of products. This format is logically developed from the IBM system 34 format (a standard format for 8 inch disks), and is specifically designed to obtain the optimum number of data bytes on a 3.5 inch MicroFloppy disk.

The format employs double density MFM coding with 512 bytes per sector and 9 sectors per track. The number of tracks per side for a double-sided disk is 80. (The single-sided disks are 70 track).

A brief description of the disk format is included at the end of this chapter (under the heading Track Format). This should be read now, if the reader is unfamiliar with the method of recording data on disks.

Control connections between the interface and the drive is made internally within the Systems Unit. The Floppy disk drive connector is a 26-way male IDC terminal mounted on the component side of the Systems Board. An associated ribbon cable assembly links the interface to the drive.

Regulated power supply voltages for the disk drive are supplied directly from the Systems Board via a Molex connector and a 4-wire cable assembly.

Details

General

The Floppy Disk Interface consists of a Western Digital WD2797-02 Floppy Disk Controller (FDC), a series of buffers, a decoding circuit for selecting and engaging the disk drive heads, and the interface connector to the disk drive.

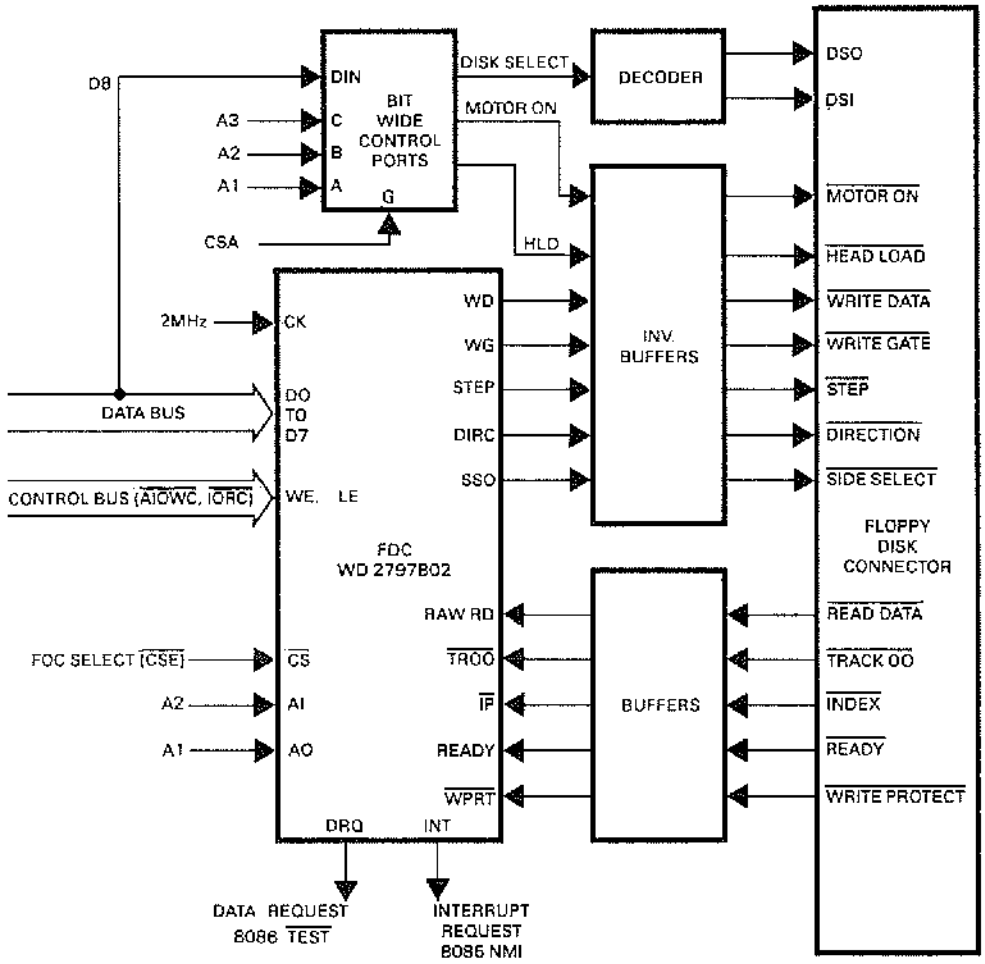


Figure 1. Floppy disk Interface

FDC Pin Definition

WD	Write Data	DRQ	Data Request
WG	Write Gate	INTRQ	Interrupt Request
STEP	Step pulse output	CLK	Clock input
DIRC	Direction control	D0 to D7	Data bus
SSO	Side Select Output	$\overline{WE}$	Write Enable
RAW RD	Read Data	$\overline{RE}$	Read Enable
$\overline{TR00}$	Track 00	$\overline{CS}$	Chip Select
$\overline{IP}$	Index Pulse	A0, A1	Register selects
READY	Ready input	MR	Master Reset
$\overline{WPRT}$	Write Protect		

Four of the control lines to the Disk Drive are wired to a latch and are under direct control of the system software.

Two of these control lines (DS0, DS1) are routed via a simple decoding circuit to select the required disk drive in a dual disk system. The third control line is the Head Load signal (HLD) for engaging the read/write head of the selected disk drive. The fourth is the motor control line (Motor On).

A full description of their use is provided in the section Programming Considerations, later in this chapter.

The remaining control and data transfer functions are implemented by the FDC. It controls the movement of the read/write head, transfers data to and from the disks, and monitors status signals from the drive.

These functions are described in the section FDC Detail, and the associated interface connections are tabulated in the section Interface Connections.

All disk transfer operations performed by the FDC (formatting, reading disk data, writing data onto disks) are initiated by the 8086 CPU. The FDC then assumes control of the data transfer.

To illustrate some of the basic principles involved, the following paragraphs present a simplified description of disk write, disk read, disk formatting, and read/write head positioning.

Disk Write

Disk write operations are initiated by the CPU issuing a write command byte over the data bus to the FDC's Command Register. The FDC then searches the disk for the location of the data (correct side, track and sector, and with the correct CRC byte values in the ID field).

When it finds the required location, the FDC asserts DRQ (Data Request) to signify to the CPU that data is required. The CPU polls this signal by monitoring its TEST input. When it detects an active DRQ signal, the CPU writes the first data byte into the FDC's Data Register and then restarts the polling of it's TEST input.

The disk write cycle then repeats for each of the bytes in the data transfer, as follows:

1. The FDC transfers the contents of its Data Register to an encoding circuit and re-asserts DRQ (Data Request).
2. The encoding circuit converts the data byte into MFM encoded data, and writes this data to disk.
3. The CPU detects the active DRQ signal and writes the next byte to the FDC Data Register.

After the last data byte in the sector is written onto the disk, a two-byte CRC is computed internally within the FDC, which is then also written onto the disk.

On completing a write cycle (completed write operation to a single sector - 512 byte transfer), the FDC asserts INTRQ (Interrupt Request). This is wired to the NMI (Non-Maskable Interrupt) input line on the CPU and informs it that the transfer cycle has finished.

The CPU can then check the FDC Status Register to see whether any errors occurred during the write operation. Reading the Status Register (or writing to the Command Register) automatically resets INTRQ, ready for the next command cycle.

Disk Read

Disk Read operations are initiated by the CPU issuing a read command byte over the data bus to the FDC Command Register. The FDC then searches the disk for the location of the data (with correct side, track and sector, and with the correct CRC byte values in the ID field).

When it finds the required location, the following repetitive disk read cycle is carried out:

1. MFM encoded disk data is read in, decoded and then assembled into a parallel byte, before being transferred to the FDC Data Register.
2. The FDC then asserts DRQ active.
3. The CPU polls for the DRQ signal on its TEST input line. When it detects an active DRQ, it reads the assembled byte from the FDC Data Register.

On completing the read cycle (completed read operation of a single sector - 512 byte transfer), the FDC asserts INTRQ (Interrupt Request). This is wired to the NMI (Non-Maskable Interrupt) input line on the CPU and informs it that the transfer cycle has finished.

The CPU can then check the FDC Status Register to see whether any errors occurred during the read operation.

Reading the Status Register (or writing to the Command Register) automatically resets INTRQ ready for the next command cycle.

Disk Formatting

Disk formatting is a similar process to disk write operations but involves transferring both data and gap information (unintelligent information used to separate areas of data on the disk, see Figure 2) onto the disk, and is executed on a track-by-track basis.

The process is initiated by the CPU issuing a Write Track command to the FDC. The FDC responds by signifying to the CPU that a byte is required by asserting DRQ. The CPU again polls for the DRQ signal on its TEST input line.

When the CPU detects an active DRQ, it writes the first gap byte into the holding register (Data Register) within the FDC. The FDC resets the DRQ output and then waits for the Index Pulse (the marker for the start of a track) to be detected.

On detecting the Index Pulse, the FDC transfers the byte within the Data Register to an encoding circuit and re-asserts DRQ. The CPU responds by writing the next byte to the Data Register and the process is repeated. The encoding circuit converts the byte into MFM data and writes the data onto the disk.

This process continues until the FDC detects the next Index Pulse, which causes the FDC to generate an interrupt request on the INTRQ output.

The interrupt request again has the same functions as described before; i.e. it indicates to the CPU the end of the command cycle. Any errors produced during the transfer can then be checked by the CPU analysing the FDC status register.

Resetting the INTRQ output is achieved by reading the Status Register or issuing a command to the FDC.

Read/Write Head Positioning

Prior to performing any of the disk transfer routines described above, the read/write head has to be positioned over the required track on the selected disk and the head load control signal issued to engage the read/write head. The head positioning operation is achieved under software control by issuing a command byte to the FDC.

The FDC responds to five different head positioning command bytes; each moves the head to a position specified by the command. The five commands are Restore, Seek, Step, Step-in and Step-out. The function of each command is detailed in the table below.

Head Positioning Commands

Command	Function
Restore	Positions the head over Track 0 on the Disk.
Seek	Positions the head to the track number specified in the Data Register.
Step	Moves the head to an adjacent track in the same direction as the previous head positioning command.
Step-in	Moves the head to the adjacent track in the direction away from Track 0.
Step-out	Moves the head to the adjacent track in the direction towards Track 0.

On completion of a head positioning command, the FDC generates an interrupt request on the INTRQ output to indicate to the CPU the end of the command cycle. Reading the Status Register or writing a new command to the FDC resets the INTRQ output.

An optional feature of each of the five commands is to automatically verify the track position of the head, by comparing the track number stored within an internal register (Track Register) with the track number contained in the ID fields on the disk.

The verification operation also checks the ID field for errors utilising the ID field CRC bytes. Failure to detect the same track number or incorrect CRC detection, sets error status bits within the Status Register. Since this feature involves reading the disk ID field, the head has to be loaded prior to the command.

The signals supplied by the FDC to the disk drive, to position the read/write head are the STEP and DIRC (direction) outputs. A step pulse with a duration of 2 μs is issued to the drive every time the FDC wants to move the head by one track location.

The FDC determines the direction of movement implied by the command and sets the state of the direction output accordingly. (Logic high to move the head away from Track 0, logic low to move the head towards Track 0).

FDC Detail

General

The FDC can be divided from a descriptive point of view into three areas of circuitry, a processor interface, a disk drive controller, and a series of registers.

The processor interface handles the transfer of data, commands and status, between the internal registers and the CPU, and also generates the interrupt and DRQ signals.

The disk drive controller responds to commands from the CPU, providing the neccessary circuitry, and input and output lines for:

1. Controlling the positioning of the drive read/write head.
2. Transferring data to and from the disks.
3. Monitoring the disk drive status.

The internal registers provide the means of exchanging information (commands, status, positional data) between the disk drive controller and the processor interface.

Processor Interface

The connections to the processor interface are detailed in the table "System Connections" at the end of the chapter. The majority of the circuitry is a straightforward interface between the 8-bit bi-directional data bus and the series of addressable registers located within the FDC.

The system software views the registers as a series of peripheral ports located in the system input/output space. The port address locations as defined by the FDC select and the system address bus connections, are detailed below. The Data, Sector and Track Registers can be written to and read from. The Command Register can only be written to, and the Status Register can only be read from.

Address	Register	Transfer
40H	Command	Write only
40H	Status	Read only
42H	Track	Read/Write
44H	Sector	Read/Write
46H	Data	Read/Write

The remaining circuitry of the processor interface consists of the control section, which produces the INTRQ and DRQ outputs.

Data Requests

The DRQ output is activated (set to logic high), during data transfer operations to and from the disk, and follows the state of an associated control bit within the Status Register. During disk read operations, the DRQ output is set active when the FDC has data available from the disk within the Data Register. The output is reset to the inactive state when the byte is read by the CPU.

During data write and formatting operations, the DRQ output is set active when the FDC Data Register is empty, and the FDC requires another byte from the CPU. The output is reset when the Data Register is loaded with a new byte.

Interrupt Requests

The INTRQ output is activated (set to logic high) on the successful completion of disk read, write, or formatting operations. It is automatically reset following these operations, on reading the Status Register or issuing a new command to the Command Register.

The INTRQ output is also set to logic high, for a variety of other conditions (premature termination of a disk transfer command sequence due to an error condition, completion of a read/write head position command, etc.).

In all cases, INTRQ can be reset by either reading the Status Register or issuing a command to the Command Register. These other conditions are detailed in the following paragraphs.

The first operation performed by the FDC on receiving a disk read or disk write command is to check whether the disk is ready for a transfer operation by analysing the READY input from the disk drive. If the input is set to logic high, the command sequence is initiated. If the input is set to logic low, the command sequence is immediately aborted, the Not Ready bit set within the Status Register, and the INTRQ output activated. The FDC also analyses the Write Protect input ($\overline{\text{WPRT}}$) from the disk drive on receiving a disk write or formatting command. If the $\overline{\text{WPRT}}$ input is activated (logic low, indicating that the disk is write protected), the write operation is terminated, INTRQ is activated and the Write Protect bit set within the Status Register.

Prior to writing data to or reading data from the disk, the FDC locates the correct sector for the transfer operation, by analysing the ID fields on the disk. Failure to detect an ID field with the correct track number, correct side number, correct sector number and correct CRC within five revolutions of the Disk, sets the Record Not Found (RNF) bit in the Status Register, activates the INTRQ output and terminates the transfer operation.

At the start of disk write and formatting operations, the FDC signifies to the CPU that the first byte is required, via the DRQ output. If during formatting, the CPU fails to supply the first byte before the FDC detects the Index Pulse, the Lost Data (LD) bit is set within the Status Register, INTRQ is activated and the formatting operation is terminated.

If during disk writes, the CPU fails to supply the first byte before the start of the data field, the same process occurs; the Lost Data bit is set, INTRQ is activated and the operation is terminated.

During disk read operations, the FDC checks the two-byte CRC code at the end of the sector data field to ensure the validity of the data. If a CRC error is present, the CRC error bit in the Status Register is set, INTRQ is activated and the read operation is terminated.

In addition to the transfer commands, the INTRQ output is also activated at the end of the command sequence for positioning the read/write head of the drives. The success of the operation is again signified by the status bits within the Status Register.

The first operation performed by the FDC on receiving the Restore command is a check of the Track 0 input ($\overline{\text{TR00}}$). If the $\overline{\text{TR00}}$ input is set low (indicating that the head is already positioned over the first track), and the verification option is not selected, the FDC clears the Track Register to zero and activates the INTRQ output.

If the $\overline{\text{TR00}}$ input is high, the FDC issues step pulses until the $\overline{\text{TR00}}$ input is set low, which then produces the same effect as described above, providing the verification option is not selected (i.e. INTRQ activated, Track Register cleared). If the $\overline{\text{TR00}}$ input is not set low within 255 step pulses, the operation is aborted, INTRQ is activated, and the Seek Error bit with the Status Register is set.

If the verification option is selected with any of the head positioning commands, the FDC moves the head to the specified position and then reads the first encountered ID field on the disk. The track number from the ID field is compared with the track number stored in the Track Register. Providing the two numbers are identical and the ID field CRC bytes are correct, the track position is deemed true, and the INTRQ output is activated.

If the track numbers match, but the CRC is incorrect, the CRC error bit within the Status Register is set and the next encountered ID field is analysed. If the FDC fails to detect an ID field with a matching track number and correct CRC within 5 revolutions of the disk, the operation is aborted, INTRQ is activated, and the Seek Error bit within the Status Register is set.

The FDC can be also issued with a command (Force Interrupt), which sets the Status Register to monitor the state of the input status lines from the drive, and causes the INTRQ output to be activated for any of the conditions detailed below:

1. Immediately the command is received.
2. Every time an Index Pulse is detected.
3. On detecting a transition on the Ready input.

Disk Drive Control

The disk drive controller circuitry acts as the interface between the disk drives and the other areas of circuitry within the FDC and consists of the disk data encoding and decoding sections, the head positioning control section and a status monitoring section.

Connections to and from the FDC disk controller circuitry are detailed in the table "Disk Drive Connections" at the end of the chapter.

At the start of a data transfer operation to the disk (write operation), the Write Gate output is activated and the CPU begins the process of transferring data bytes to the Data Register in parallel format.

The FDC transfers each byte from the Data Register to the encoding circuit. The encoding circuit converts the bytes into MFM double density encoded data, which is then supplied to the disk via the Write Data output.

When writing to tracks with a track number greater than 43, the encoding circuit provides automatic write precompensation. The precompensation value is set by a potentiometer (VR1), located on the Systems Board.

The decoding section of the disk controller circuitry decodes the MFM encoded data from the Raw Read input, during transfer operations from the disk. The decoder is a phase-locked loop data separator circuit based around an internal voltage controlled oscillator (VCO) and phase detector.

The centre frequency of the VCO is set by a variable capacitor (VC1), located on the Systems Board. A second variable control VR2 sets the read window pulse width.

The head positioning control section responds to the head positioning commands supplied to the Command Register and controls the STEP and Direction (DIRC) outputs.

The rate at which the 2 μs step pulses are issued to the drive, to move the head from track-to-track is specified by the command. Issuing a step pulse to the drive to move the head towards Track 0, automatically decrements the Track Register. Issuing a step pulse to the drive to move the head away from Track 0, automatically increments the Track Register.

The status monitoring section monitors the logic state on the four inputs from the disk drive, READY, $\overline{\text{TR00}}$ Index Pulse ($\overline{\text{IP}}$) and Write Protect ($\overline{\text{WPRT}}$). All the four inputs can be monitored by issuing any of the head positioning commands or the Force Interrupt command, and then examining the contents of the Status Register.

The effect of the status inputs on the operation of the FDC is dependent on the command issued to the Command Register and the logic state of the input line.

Command Register

The 8-bit Command Register holds the command supplied from the CPU which determines the type of operation carried out by the FDC, and is located at address location 40H in the I/O space. The register can only be written to with one of eleven predefined command words. A command currently in progress is indicated by an associated control bit within the Status Register.

The eleven commands can be divided into four different categories as detailed below. A detailed description of each command is provided in the Programming Considerations section.

Type 1. Read/write head positioning commands: Restore, Seek, Step, Step-in, Step-out.

Type 2. Data transfer commands: Read Sector, Write Sector.

Type 3. Format code transfer commands: Read Address, Read Track, Write Track.

Type 4. Interrupt command: Force Interrupt.

Status Register

The 8-bit Status Register holds status information, which is dependent on the command operation performed by the FDC. The register is located at address 40H in the system I/O space, and can only be read from. Some of the bits within the register signify the state of the control inputs from the disk drive, whilst others indicate the status of the command operation.

Track Register

The 8-bit Track Register indicates the track number of the position of the drive read/write head and is located at address location 42H in the system I/O space. The register can be written to and read from. The FDC updates the track register during head positioning command operations, every time the drive head is moved to an adjacent track.

Sector Register

The 8-bit Sector Register is used to store a sector number, which indicates to the FDC the desired location on the disk for a transfer operation. The register is located at address location 44H in the system input/output space and can be written to and read from.

Data Register

The 8-bit Data Register is the holding register during transfer operations to and from the disk, and is located at address location 46H in the system I/O space. The register performs a different function during the Seek Command, when the register is programmed with the desired track location.

Programming Considerations

General

The system software controls the Floppy Disk Interface directly, by addressing control ports in the System Input/Output Space located within the FDC and a bit-addressable latch.

The FDC also communicates with the system software via two output lines to the CPU:

DRQ Data Request indicates that the FDC is waiting for a byte transfer to or from its Data Register.

INTRQ Interrupt Request indicates one of the following:

1. Command terminated successfully.
2. Command terminated unsuccessfully.
3. Force Interrupt command executed.

The disk control functions performed directly by writing to the latch are listed below.

1. Disk Drive Selection.
2. Disk Drive Motor Control.
3. Head Loading.

The functions performed using FDC commands are listed below:

1. Head positioning.
2. Data transfers.
3. Disk formatting.
4. Status Monitoring.

These operations are described in the rest of this section, including a description of how the FDC Status Register is affected by each command.

Disk Drive Selection

Selecting the integral disk drive for operation is achieved by writing data to a bit wide port mapped into the system I/O space at address location 01H. The port is connected to the LSB data line of the high order section of the data bus (D8). Setting the port output to logic high, selects the integral disk drive for operation (i.e. writing a value of FFH to I/O address 01H).

The Sony MicroFloppy Disk drive is selected by setting the logic states on the two drive select outputs (DS0 and DS1), to match a switch setting located on the drive.

Writing FFH to I/O address 01H matches the select state of the drive when it is configured as Drive 2. This is the normal configuration switch setting of the integral disk drive. (the drive can be designated as Drive 1, 2, 3 or 4 according to the position of the switch - in theory allowing four drives to be integrated into a single system).

Writing 00H to I/O address 01H deselects drive 2 and matches the select state of a drive configured as Drive 3. This extra select state is provided to cater for any future upgrade of the F1 into a dual disk system.

The two drive select output lines (DS0 and DS1) form a two-bit code which can be encoded into one of two different values only since DS0 is always the inverse of DS1. Writing FFH to 01H sets DS0 to logic low and DS1 to logic high. Writing 00H to 01H produces the inverse condition on DS0 and DS1.

Motor Control

A motor control signal (Motor On) is provided to allow the disk drive motor to be switched on and off. Controlling the motor is achieved by writing data to a bit wide port mapped into the system I/O space at address location 05H. The port is connected to the LSB data line of the high order section of the data bus (D8). Setting the port output to logic high, switches the disk drive motor on (i.e. writing a value of FFH to I/O address 05H).

The double-sided Sony disk drives take approximately 1.6 seconds for the drive to reach its correct rotational speed, following switch on. Setting the port output to logic low, switches the motor off (i.e. writing a value of 00H to I/O address 05H).

Head Loading

Loading the read/write head of the selected disk drive is achieved by writing data to another bit wide port. This is mapped at 03H in the system I/O space and is also connected to the LSB data line of the high order section of the data bus (D8). Setting the port output to logic high engages the read/write heads (i.e. writing FFH to I/O address 03H).

The time taken for the Sony double-sided disk drive to engage the read/write head, after the head load signal is set active is the order of 60 ms. A 60 ms delay should therefore be implemented after issuing the head load signal before performing disk data transfer operations.

The port output also controls the DISC LED indicator located on the front panel. Everytime the read/write heads are loaded, the indicator is switched on. Unloading the heads (writing 00H to I/O address 03H) switches the indicator off.

Head Positioning

Positioning the read/write head of the selected disk drive is achieved by issuing one of the five Type 1 commands to the Command Register of the FDC. Termination of a command (successful or otherwise) is signified by an interrupt request to the CPU. The format of each of the head positioning commands is detailed below.

Type 1 Commands

D7							D0	Command
0	0	0	0	1	V	r1	r0	Restore
0	0	0	1	1	V	r1	r0	Seek
0	0	1	T	1	V	r1	r0	Step
0	1	0	T	1	V	r1	r0	Step-in
0	1	1	T	1	V	r1	r0	Step-out

T = Track Update Flag
V = Verify Flag
r1, r0 = Stepping motor rate

Each of the commands contain a Verify Flag bit and stepping motor rate bits. The stepping rate bits have to be set according to the track-to-track access time of the disk drive. The combination of bits allow four different stepping motor rates to be selected as detailed below. The track-to-track access time of the Sony double-sided disk drive is 12 ms.

r1	r0	Rate
0	0	3 ms
0	1	6 ms
1	0	10 ms
1	1	15 ms

The Verify Flag bit determines whether the FDC verifies the track position after positioning the head, by reading the sector ID fields on the disk. Verification is performed, when the Verify Flag bit is set to logic high, and requires the head to be loaded prior to the command.

The verification operation checks the track number in the sector ID field with the number contained in the Track Register, and also checks the ID field CRC character.

Failure to match the track number or failure to find a matching track with a valid CRC, within five revolutions of the disk, causes the Seek Error bit to be set in the Status Register. If the verification operation detects a CRC error within any of the ID fields checked, the CRC error bit is set within the Status Register.

The Step commands contain a Track Update Flag, which determines whether the Track Register is updated every time the head moves to an adjacent track. The Track Register is updated if the Track Update Flag is set to logic high.

Restore Command

The Restore command is used to position the head over Track 0 of the disk. The FDC will issue up to 255 step pulses at the rate specified by the stepping rate bits, in an attempt to locate the first track on the disk.

Failure to locate Track 0 (by monitoring the state of the $\overline{\text{TR00}}$ input) within 255 step pulses, causes the command to terminate, and the Seek Error bit to be set within the Status Register. If the Verify Flag bit is set, verification of the track position is carried out as detailed above.

Seek Command

Prior to issuing the Seek command, the Data Register has to be loaded with the desired track number and the Track Register is presumed to contain the current position of the head. On receiving the command, the FDC sets the DIRC output to move the head in the direction of the desired track.

Step pulses are then issued at the rate specified by the stepping rate bits, and the Track Register updated on each pulse, until the number in the Track Register coincides with the number in the Data Register. If the Verify Flag is set, verification of the track position is carried out.

Step Command

Issuing a Step command moves the head one track location in the direction specified by the previous command. If the Track Update Flag is set, the Track Register is updated following the issue of the Step pulse.

If the Verify Flag is set, verification of the track position is carried out, after a delay determined by the stepping rate bits (i.e. A delay of 12 ms for the Sony double-sided drive).

Step-in Command

Issuing the Step-in command moves the head one track location away from Track 0. If the Track Update Flag is set, the Track Register is incremented following the issue of the Step pulse.

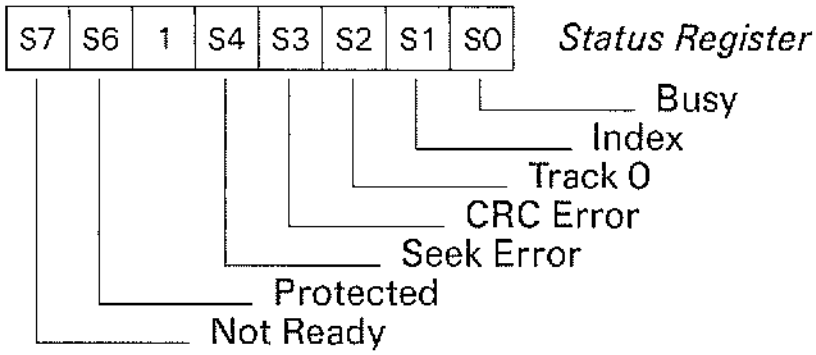
If the Verify Flag is set, verification of the track position is carried out, after a delay determined by the stepping rate bits.

Step-out Command

Issuing the Step-out command moves the head one track location towards Track 0. If the Track Update Flag is set, the Track Register is decremented following the issue of the Step pulse. The Verify Flag functions in the same manner as described above for the Step and Step-in commands.

Status Register

The format of the Status Register following a head positioning command is as detailed below. On issuing a command to the Command Register, the FDC resets the Status Register to monitor certain status conditions implied by the new command. Due to internal timing delays, the Status Register does not contain valid status information, until 14 μs after the new command is issued.



Busy	A logic high on Busy indicates that the command is in progress; logic low indicates that the command sequence is complete.
Index	Inverted copy of the Index Pulse input from the disk drive. Logic low when the pulse occurs.
Track 0	Inverted copy of the $\overline{\text{TR00}}$ input from the drive. Logic high indicates that the head is positioned over Track 0.
CRC Error	Logic high indicates a CRC error was detected within a sector ID field during verification of the track.
Seek Error	Logic high indicates a matching track number was not located within a sector ID field during the verification operation. The bit is also set high if the FDC fails to detect a logic low on the $\overline{\text{TR00}}$ input following a Restore command.
Protected	Inverted copy of the $\overline{\text{WPRT}}$ input from the drive. Logic high indicates that the selected disk drive contains a write protected disk.
Not Ready	Inverted copy of the READY input from the drive. Logic high indicates that the drive is not ready for a data transfer operation.

Data Transfers

Transfer of data to and from the disk is controlled by the two Type 2 commands, Write Sector and Read Sector. Prior to issuing the command to the Command Register, the Sector Register must be loaded with the desired sector number, to determine the source/destination of the data.

Termination of the command (successful or otherwise) is signified by an interrupt request to the CPU (NMI). Both command operations are prematurely terminated, if the disk drive is not ready for the transfer operation, as indicated by the READY input.

The format of the data transfer commands is detailed below.

Type 2 Commands

D7							D0	Command
1	0	0	m	1	1	U	0	Read Sector
1	0	1	m	1	1	U	0	Write Sector

m = Multiple Record Flag
U = Update SSO

Both commands contain a Multiple Record Flag bit and an Update SSO bit. The Update SSO bit is used to select the disk side for the data transfer operations and affects the logic state of the Side Select Output (SSO), supplied to the disk drive. When U is set to logic low, the SSO is updated to logic low (side 0). When U is set to logic high, SSO is updated to logic high (side 1).

The Multiple Record Flag bit selects whether data transfers are from/to a single sector or multiple sectors within a track. Single sector transfers are initiated when the m Flag is set to logic low, multiple sector transfers when the m Flag is set to logic high.

Write Sector

On receipt of the Write Sector command, the FDC begins the process of searching the sector ID fields of the track for the desired destination for the data.

When an ID field is found with the correct track number (as specified by the Track Register), the correct side number (as specified by the U bit in the command), the correct sector number (as specified by the Sector Register), and correct CRC character; the FDC generates an active state on DRQ, which informs the CPU to write the first data byte into the Data Register.

If an ID field is not found containing the correct information within five revolutions of the disk, the command is aborted and the Record Not Found bit in the Status Register set.

If any of the ID fields encountered, contain an incorrect CRC character, this is also recorded in the Status Register.

On receipt of the first data byte, the FDC activates the Write Gate output and on detecting the start of the data field, writes the data byte to the disk. The process then continues with the FDC asserting DRQ every time a new byte of data is required.

After the 512th data byte is written to the disk, a two-byte CRC character is automatically generated and written onto the disk. If the data written to the sector only fills a part of the data field, the remaining area should be programmed with a series of zeroes, to complete the whole data field.

If a single sector write operation was specified by the Write Sector command, the Write Gate output is then deactivated and the command operation terminated.

If the Write Sector command specified a multiple sector transfer, the Sector Register is incremented and the process repeated, starting from the verification operation on the next ID field.

The multiple sector transfer operation continues until terminated either by issuing a Force Interrupt command, or after the sector number is incremented to a value exceeding the number of sectors on the track. In the latter case, the FDC automatically terminates the command after five revolutions of the disk, since the verification of the ID field will not be able to locate a matching sector number.

Due to the unpredictability of the FDC terminating the transfer in a "clean" manner, the usage of multiple sector transfer operations as dictated by the FDC are of limited value. For this reason and the fact that the majority of applications only generally request single sector transfers, the BIOS only supports single sector operations.

Fast "multiple" single sector transfers are supported by using a sector interleave factor of 2. (Interleaving is the term given to altering the physical order of sectors within a track to improve the access time when transferring more than one sector).

Failure of the CPU to write the first data byte to the Data Register before the arrival of the sector data field causes the FDC to, set the Lost Data bit in the Status Register, activate the INTRQ output, and abort the command.

Failure of the CPU to supply a data byte to the Data Register on receiving a DRQ request after the first byte (i.e. within 11.5 μs), causes the FDC to write a byte of zeroes onto the disk and also set the Lost Data bit, but does not terminate the command sequence.

Read Sector

On receipt of the Read Sector command, the FDC begins the process of searching the sector ID fields of the track for the desired source of data.

When an ID field is found with the correct track number (as specified by the Track Register), the correct side number (as specified by the U bit in the command), the correct sector number (as specified by the Sector Register), and the correct CRC character; the FDC reads the data bytes from the following data field, and informs the CPU, via the DRQ output, every time a data byte is stored in the Data Register.

If an ID field is not found containing the correct information within five revolutions of the disk, the command is aborted and the Record Not Found bit in the Status Register set.

If any of the ID fields encountered, contain an incorrect CRC character, this is also recorded in the Status Register.

Failure of the CPU to read the contents of the Data Register before it is overwritten with the next byte of data from the disk, causes the FDC to set the Lost Data bit in the Status Register.

If a single sector read operation was specified by the Read Sector command, the sequence terminates with the FDC testing the CRC character at the end of the data field. If the CRC character is incorrect, the CRC Error bit is set within the Status Register.

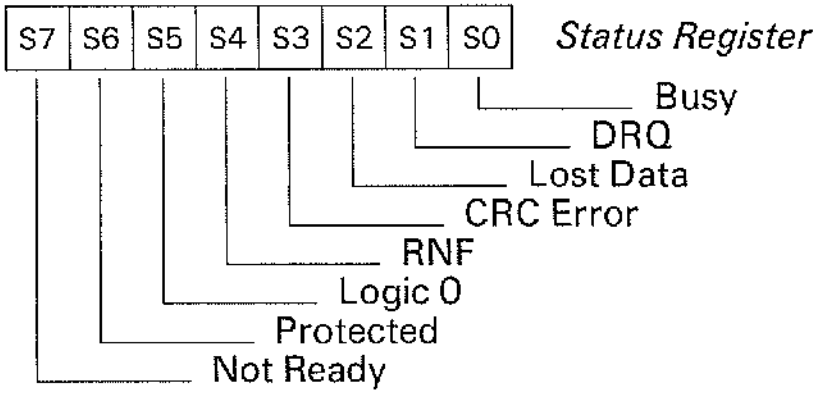
If the Read sector command specified a multiple sector transfer, the Sector Register is incremented and the process repeated for the next sector, providing the CRC character tested at the end of the previous data field was true. If the CRC character was incorrect, the multiple sector routine is prematurely aborted.

If no CRC Errors are detected, the multiple sector operation continues until terminated either by issuing a Force Interrupt command, or after the sector number is incremented to a value exceeding the number of sectors on the track. In the latter case, the FDC automatically terminates the command after five revolutions of the disk, since the verification of the ID field will not be able to locate a matching sector number.

The BIOS does not support multiple sector transfers (see Write Sector section above).

Status Register

The format of the Status Register following a data transfer command is as detailed below. On issuing a command to the Command Register, the FDC resets the Status Register to monitor certain status conditions implied by the new command. Due to internal timing delays, the Status Register does not contain valid status information, until 14 μs after the new command is issued.



Busy	A logic high on Busy indicates that the command is in progress; logic low indicates that the command sequence is complete.
DRQ	A copy of the DRQ output to the CPU. A logic high indicates that the FDC has data available during a read operation, and requires data during a write operation.
Lost Data	Logic high indicates that the CPU did not respond to the DRQ output in time, and as a result; valid data was not written onto the disk during a write operation, valid data was not read from the disk during a read operation.
CRC Error	During the write operation, a logic high indicates a CRC error was detected within one or more sector ID fields. During the read operation, a logic high indicates a CRC error was detected either within one or more ID fields, or a CRC error detected at the end of a data field.
RNF	Record Not Found. A logic high indicates that the transfer operation was unable to locate an ID field containing the correct side, track and sector number, with a valid CRC character.
Protected	Inverted copy of the $\overline{\text{WPRT}}$ input from the drive, during write operations. Logic high indicates that the selected disk drive contains a write protected disk. During read operations set to logic low.
Not Ready	Inverted copy of the READY input from the drive. Logic high indicates that the drive is not ready for a data transfer operation.

Formatting Commands

The formatting commands are the three Type 3 commands, Write Track, Read Track and Read Address. Write Track is the command for formatting disks. The two other commands enable the programmer to check the disk format.

Termination of the command is signified by an interrupt request on the NMI input of the CPU. The command operations are terminated prematurely, if the disk drive is not ready for the transfer operation as indicated by the READY input.

The format of these commands is detailed below.

Type 3 Commands

D7							D0	Command
1	1	0	0	0	1	U	0	Read Address
1	1	1	0	0	1	U	0	Read Track
1	1	1	1	0	1	U	0	Write Track

U = Update SSO

Each of the three commands contain an Update SSO bit. This bit is used to select the disk side for the formatting operations and affects the logic state of the Side Select Output (SSO), supplied to the disk drive. When U is set to logic low, SSO is updated to logic low (side 0). When U is set to logic high, SSO is updated to logic high (side 1).

Track Format

The track format used on the MicroFloppy disks is illustrated at the end of the chapter.

Read Address

On receipt of the Read Address command, the FDC searches the disk for an ID field. The first ID field encountered is then read from the disk. The FDC transfers the six ID field bytes detailed below, from the disk to the Data Register one byte a time. Every time a data byte is stored in the Data Register, the FDC generates DRQ. The ID field track number is also transferred to the Sector Register.

Byte	Description
1	Track Number
2	Side Number
3	Sector Number
4	Sector Length (02H)
5	CRC 1
6	CRC 2

If the FDC fails to locate an ID field within five revolutions of the disk, the command is aborted and the Record Not Found bit in the Status Register set. If the ID field contains an incorrect CRC character, this is also recorded in the Status Register.

Failure of the CPU to read the contents of the Data Register, before the register is overwritten with the next byte of data from the disk, causes the FDC to set the Lost Data bit in the Status Register.

Read Track

The Read Track command allows a complete track of information (Gaps, Headers and Data bytes) to be read from the disk. On receipt of the Read Track command, the FDC monitors the logic state on the Index Pulse input. On detecting the leading edge of the pulse, the FDC transfers all the Gap, Header and Data bytes from the track into the Data Register on a byte-by-byte basis.

Every time a byte is transferred into the Data Register, the FDC informs the CPU that a byte is available, via the DRQ line. The command terminates after one full revolution of the disk, on detecting the Index Pulse again. No CRC error checking is carried out.

Every time the FDC detects the ID field and Data field address marks, it synchronises the internal decoding circuitry, to ensure that all information following each sector ID address mark is valid. Since the FDC might not be synchronised to the gap information previous to the ID address mark, this information is not guaranteed to be valid.

Failure of the CPU to read the contents of the Data Register, before the register is overwritten with the next byte of data from the disk, causes the FDC to set the Lost Data bit in the Status Register.

Write Track

The Write Track is the command used to format the tracks on the disk. All data and gap information is supplied to the disk by building a image of the track in memory as detailed on the next page, and transferring the format data to the Data Register.

On receipt of the Write Track command, the FDC generates DRQ, to inform the CPU to write the first byte from the track image into the Data Register.

On detecting the leading edge of the Index Pulse, the FDC transfers the first byte to the disk and requests another byte from the CPU. The process then continues with the CPU supplying the format bytes with the FDC generating DRQ every time a new byte is required.

The command sequence terminates after one full revolution of the disk, on detecting the leading edge of the next Index Pulse.

Track Image

Number of bytes required	Byte Value (Hex)	Description	
80	4E	Gap	
12	00	Sync	
3	F6	Control Bytes	*PREAMBLE*
1	FC	Index Mark	
50	4E	Gap	
12	00	Sync	
3	F5	Control Bytes	
1	FE	ID Address Mark	
1	00 to 50*	Track Number	
1	00 or 01	Side Number	
1	01 to 09	Sector Number	*SECTOR*
1	02	Sector Length	(Repeated 9
1	F7	CRC character command	times
22	4E	Gap	changing
12	00	Sync	the Sector
3	F5	Control Bytes	Number each
1	FB	Data Address Mark	time)
512	00	Data Bytes	
1	F7	CRC character command	
84	4E	Gap	
598**	4E	Gap	*POSTAMBLE*

* For 80 track disks.

** Nominal figure, write operation continues until terminated by the INTRQ output, on detecting the Index Pulse.

Some of the bytes supplied to the Data Register are not written onto the disk, but act as control bytes to the FDC. These are the three bytes F5H, F6H and F7H.

F5H commands the FDC to perform two functions; write a unique byte pattern corresponding to A1 with a missing clock transition onto the disk and initialize the CRC generator circuitry. F6H commands the FDC to write a unique pattern corresponding to C2H with a missing clock transition onto the disk.

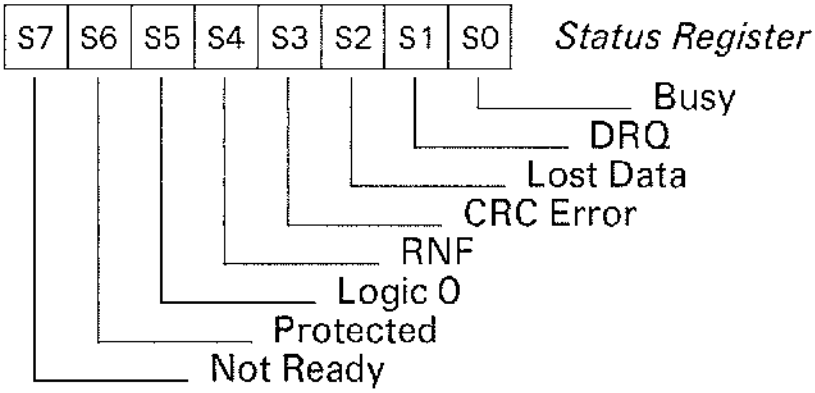
These patterns enable the FDC to interpret the byte following the three byte pattern as an address mark, during decoding operations. The byte F7H commands the FDC to write the computed two-byte CRC character onto the disk.

Failure of the CPU to write the first byte to the Data Register before the arrival of the leading edge of the Index Pulse causes the FDC to, set the Lost Data bit in the Status Register, activate the INTRQ output, and abort the command.

Failure of the CPU to supply a data byte to the Data Register on receiving DRQ after the first byte, causes the FDC to write a byte of zeroes onto the disk and also set the Lost Data bit, but does not terminate the command sequence.

Status Register

The format of the Status Register following a formatting command is as detailed below. On issuing a command to the Command Register, the FDC resets the Status Register to monitor certain status conditions implied by the new command. Due to internal timing delays, the Status Register does not contain valid status information, until 14 μs after the new command is issued.



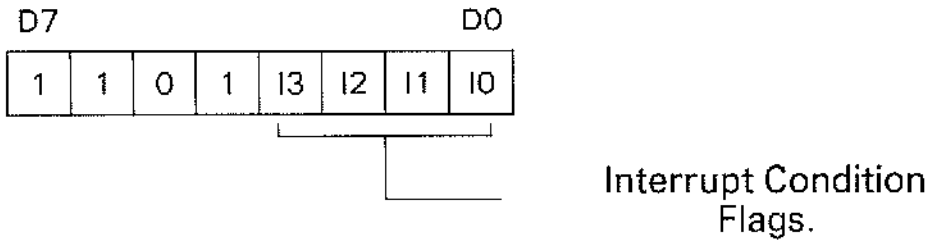
Busy	A logic high on Busy indicates that the command is in progress; logic low indicates that the command sequence is complete.
DRQ	A copy of the DRQ output to the CPU. A logic high indicates that the FDC has data available during the Read Track/Read Address operations, and requires data during a Write Track operation.
Lost Data	Logic high indicates that the CPU did not respond to the DRQ output in time, and as a result; valid data was not written onto the disk during a Write Track operation/valid data was not read from the disk during a read operation.

CRC Error	During the Read Address operation, a logic high indicates a CRC error was detected within the sector ID field. During the Read Track and Write Track operations set to logic low.
RNF	Record Not Found. A logic high indicates that the Read Address operation was unable to locate an ID field. During Read Track and Write Track set to logic low.
Protected	Inverted copy of the $\overline{\text{WPRT}}$ input from the drive, during the Write Track operation. Logic high indicates that the selected disk drive contains a write protected disk. During read operations set to logic low.
Not Ready	Inverted copy of the READY input from the drive. Logic high indicates that the drive is not ready for a data transfer operation.

Force Interrupt Command

The Force Interrupt command is a Type 4 command and is used to force the FDC to generate an interrupt via the INTRQ output on detecting a certain condition. The conditions are defined by setting certain bits within the command byte as detailed below.

Type 4 Command



Interrupt Condition Flags

I3	I2	I1	I0	Interrupt Condition
x	x	x	1	Not Ready to Ready Transition
x	x	1	x	Ready to Not Ready Transition
x	1	x	x	Every Index Pulse
1	x	x	x	Immediate Interrupt
0	0	0	0	No Interrupt

More than one condition can be set at any time by the appropriate combination of Interrupt Condition Flags.

If any of the Force Interrupt commands are issued to the Command Register when a command is already in operation, the current command is terminated and the Busy bit in the Status Register reset. All the other bits in the Status Register are left unchanged.

If the command is issued when there is no other command currently in operation, the Status Register is set to monitor the same conditions as a Type 1 command, as previously described.

The No Interrupt command functions in an identical manner to the other Force Interrupt commands, as described in the paragraph above, but does not produce an interrupt request on INTRQ.

The INTRQ output is reset following any of the Force Interrupt commands, by issuing any new command to the Command Register or by reading the Status Register. The only method available to reset the INTRQ output after issuing the Immediate Interrupt command, is to issue the No Interrupt command.

After issuing a Force Interrupt command to the Command Register:

1. Another command should not be issued for at least 8 μs, to allow the FDC to process the interrupt command.
2. The Status Register should not be read for 14 μs to ensure the register contains valid status information.

Disk Drive Connections

$\overline{DS0}$	Drive Select 0. Control signal driven by a single-bit control port mapped in the system I/O space. Active state, logic low. When active, selects the disk drive configured as drive 2.
$\overline{DS1}$	Drive Select 1. Control signal driven by a single-bit control port mapped in the system I/O space. Active state, logic low. When active, selects the disk drive configured as drive 3.
HLD	Head Load. Control signal driven by a single-bit control port mapped in the system I/O space. Active state, logic low. When active, engages the read/write head of the selected disk drive against the disk.
Motor On	Control signal driven by a single-bit control port mapped in the system I/O space. Active state, logic low. When active, starts the disk drive motor of the selected disk drive.
SSO	Side Select Output. Control signal from the FDC used to select the side 0 or side 1 of double-sided disks. Follows the state of an associated control bit within an internal register. A logic high on SSO selects side 0 and a logic low, side 1.
STEP	Step Pulse. Pulsed output generated by the FDC for positioning the disk drive read/write head. Each positive going pulse moves the head to an adjacent track location in the direction determined by the DIRC output.
DIRC	Direction Control. Control signal generated by the FDC to determine the direction of movement of the disk drive read/write head. When DIRC is set to logic high, each step pulse causes the head to step in one track (away from track 0). When DIRC is set to logic low, each step pulse causes the head to step out one track (towards track 0).

WG	Write Gate. Control signal generated by the FDC. Active state, logic high. When active, enables current to flow into the disk drive read/write head.
WD	Write Data. Output for writing MFM encoded data to the disk drive.
RAW RD	Raw Read. Input to the FDC for MFM encoded data from the disk drive.
READY	Input to the FDC from the disk drive. When set to logic high, indicates that the selected disk is ready for data transfer operations. When set to logic low, indicates that the selected disk is not available. In this condition attempted data transfer operations between the FDC and the disk drive are inhibited, and cause an active interrupt request to be generated on INTRQ. READY also sets an associated control bit within an internal register according to the logic state on the control line.
$\overline{\text{TR00}}$	Track 00. Control input from the disk drive. When $\overline{\text{TR00}}$ is set to logic low, indicates to the FDC that the read/write head of the selected drive is positioned over track 0 of the disk.
$\overline{\text{IP}}$	Index Pulse. Control input from the disk drive. A negative going pulse generated every revolution of the disk, which informs the FDC of the start of the first sector of each track.
$\overline{\text{WPRT}}$	Write Protect. Control input from the disk drive. When $\overline{\text{WPRT}}$ is set to logic low, any attempted write operation to the selected drive is inhibited. $\overline{\text{WPRT}}$ also sets an associated control bit within an internal register according to the logic state in the control line.

Track Format

Data is recorded on a disk in concentric circles, known as tracks. When a disk is inserted into the disk drive, the Auto Shutter is opened to allow the read/write head of the disk drive access to the disk surface.

When the head is loaded, the head makes physical contact with the radial slot of magnetic material exposed, when the shutter is open. Information on a track is read and written serially as the disk rotates within its protective plastic shell.

Each track of the disk is divided into nine sectors by software formatting (soft sectoring). The sectors are recorded onto each track of the disk by issuing a format command to the FDC and then writing all the bytes onto the disk as illustrated in Figure 2.

The start of each track is marked by a single index pulse, which is generated by the disk drive every revolution of the disk.

Each sector has an identification (ID) field and a data field, separated by gaps of unintelligent information.

The gaps are required to allow the FDC to process information from the disk during disk reads, and also to take into account variations in drives, such as motor speed variations.

The ID field defines the data field that follows, by specifying its track number, side number, sector number and length of the data field in bytes.

Both the ID and data fields begin with an address mark and end with a two-byte cyclic redundancy check (CRC) character for detecting errors in the previous field. Different address marks are used to distinguish between the two fields.

All the serial data on the track (ID fields, data fields and gap information) are recorded on the disk by Modified Frequency Modulation (MFM). In MFM encoding, both serial data bits and clock pulses are interleaved into the data stream.

In order to distinguish the address marks on the disk from data bytes which may be identical, the address marks are recorded with missing clock pulses in predefined locations. The address marks are the only bytes on the disk with missing clock bits, and are produced by the FDC during formatting. These are then used by the FDC for recognition and synchronisation during disk accesses.

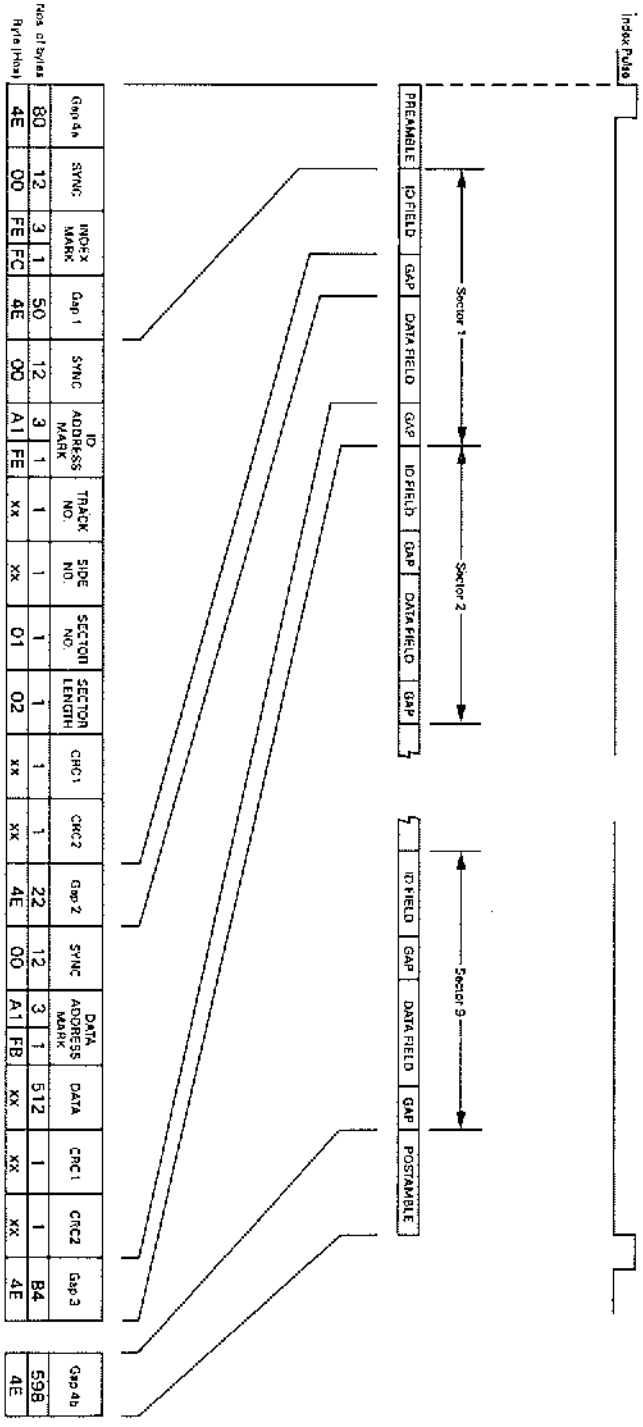


Figure 2. Track Format

Serial Interface 2.7

Contents

Introduction
Details
- General
- SIO Overview
- SIO Architecture
- Processor Interface
- Write Register Definition
- Write Register Summary
- Write Register 0
- Write Register 1
- Write Register 2
- Write Register 3
- Write Register 4
- Write Register 5
- Write Register 6
- Write Register 7
- Read Register Definition
- Read Register 0
- Read Register 1
- Read Register 2

SIO Interrupt Sequence
Keyboard/Mouse Data
Sound Generation
Channel A Programming Details
- Copy Registers
- Initialisation

RS232C Communications
- General
- RS232C Connector Detail

Channel B Programming Details
- Copy Registers
- Setting the Base Vector
- Asynchronous Communications

SIO Pin Detail
- System Connections
- Channel A Connections
- Channel B Connections

Illustrations

1. Serial Interface
2. RS232C Connector detail

Introduction

The Serial Interface is located on the System Board and consists of the elements of circuitry as illustrated on the block diagram Figure 1. The interface provides two separate serial channels; one channel for receiving data from the Infra-red (IR) Keyboard or Mouse and also for generating sound; the second channel for communication between the F1 and external equipment via an RS232C link. The Serial Interface also provides part of the system interrupt structure.

The receive channel for the Keyboard/Mouse data is programmed to operate in synchronous mode (Monosync) at a data rate determined by the incoming data stream. It is supplied with keyboard/mouse data via the IR receiver board which decodes the incoming data and converts it into an acceptable serial waveform. (Details of the IR Receiver Board are discussed in the chapter headed Systems Unit).

The transmit channel for generating sound is also programmed to operate in the synchronous mode Monosync (to match the channel requirements for the keyboard data). Sound is produced by transmitting the synchronous waveform to a loudspeaker, via a filter/amplifier circuit.

The programmer can control the frequency, waveform shape, volume, and duration of audio output, and has the facility to produce either simple tones or complex synthesized sounds. The frequency of the audio output is set by the transmit baud rate for the channel, which is supplied from the System Board Timer (the Z80 CTC).

The RS232C channel can be programmed to operate in either asynchronous or synchronous modes, with transmit and receive baud rates determined either via the Z80 CTC or via the external data communications equipment.

The RS232C interface is able to support:

1. Asynchronous communications with 5, 6, 7 or 8 bits per character and various choices of stop bits and parity sense.
2. Bit oriented synchronous communications, such as SDLC and HDLC.
3. Byte oriented synchronous communications, such as Monosync and Bisync.

The interrupt structure operated within the system is formed by the interrupt handling facilities within the Z80 SIO and the Z80 CTC. These two devices operate together to provide a prioritised interrupt structure. Both devices can generate an interrupt and also produce an interrupt vector to indicate the cause of interrupt. This is described in detail in the chapter headed "Interrupt Control".

Details

General

The Serial Interface consists of; a serial input/output controller, a data selector, a number of line drivers/receivers, a signal conditioning circuit which processes the incoming keyboard/mouse data, an amplifier and associated loudspeaker for producing sound, and a connector (25 pin D-type female connector for the RS232C channel).

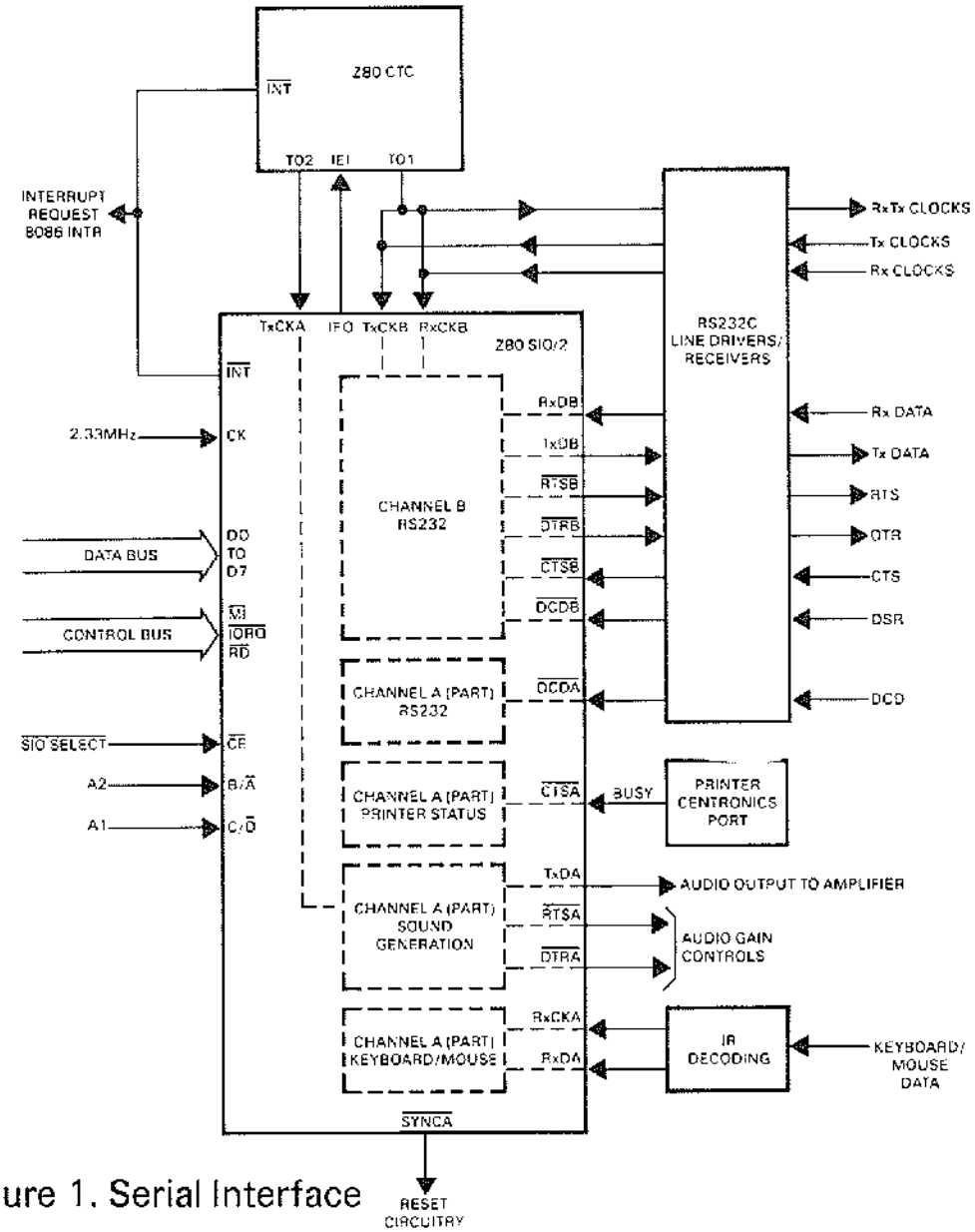


Figure 1. Serial Interface

The major circuit element of the Serial Interface is the serial input/output controller which is a Zilog Z80 SIO/2. The SIO is a programmable device which incorporates all the circuitry for interfacing between the 8086 CPU and the two serial channels. A description of each connection to the SIO is provided at the end of this chapter.

Due to the relative complexity of the SIO compared with other programmable devices on the Interface Board, a brief overview of the device is presented in the following paragraphs.

Subsequent paragraphs describe the internal structure and the utilisation of the device within the system in more detail.

SIO Overview

The SIO is a peripheral device, which performs the function of an intelligent two-channel parallel to serial/serial to parallel converter.

The two channels (designated Channel A and Channel B) are totally independent of each other and can be programmed to operate in either asynchronous or synchronous serial data communications modes.

Channel A is used as a bi-directional channel with the receive channel set for receiving synchronous data from either the keyboard or mouse and the transmit channel set for transmitting a synchronous datastream to the sound output circuit.

Channel B is available to the programmer for use as a full duplex, multi-mode RS232C link.

Both channels have to be initialised with the required serial data communications mode prior to any transmission or reception. The programmer is then able to view the SIO as two simple parallel input/output ports for the majority of data transfer operations; one bi-directional port for Channel B; one input port for channel A - Keyboard/mouse communications, one output port for channel A - sound output.

The SIO provides all the necessary facilities for organising data transfers over the two serial channels. These include:

1. Conversion of the parallel data into the correct serial data format for transmission over the selected serial channel. In asynchronous modes, this involves the automatic insertion of start, stop and parity bits, and assembling the parallel data into the correct number of serial bits per character.
2. Conversion of the input serial data from the serial channels into parallel form, and checking the incoming data for errors. In asynchronous modes, this involves stripping the start, stop and parity bits from the serial data and checking for parity and framing errors.

When operating in synchronous modes, similar automatic facilities are available within the SIO, dependent on the selected mode.

In Bisync and Monosync, pre-programmed sync characters and CRC characters can be automatically inserted during transmissions. The SIO strips sync characters from received data, and also analyses the received data for errors, utilising the received CRC characters.

In SDLC and HDLC, the SIO performs the following functions:

1. Abort sequence generation and detection.
2. Zero insertion and deletion.
3. Flag insertion between messages.
4. Address field recognition.
5. I-field residue handling.
6. CRC generation and detection.

The baud rates for the two channels are determined by clock sources external to the SIO. In asynchronous modes, the clocks can be divided internally within the SIO under program control, prior to being used to set the baud rates. This facility is not available in synchronous modes.

Each channel also provides inputs and outputs which are specifically tailored to function as modem control lines for co-ordinating serial data transfer operations. The outputs are under direct control by the programmer.

The inputs can be monitored under software control by polling, or be programmed to generate an interrupt to the CPU. These control lines can also be redefined as general purpose input and output control signals.

The SIO can also be programmed to generate interrupts to signal to the processors the status of various transmit and receive conditions in the two communication channels. These include; receive data available, transmit data required, and the detection of various error conditions.

The interrupt structure of the SIO is such that on generating an interrupt to the CPU, it can also produce an interrupt vector internally which defines the cause of the interrupt. This can then be used by the CPU to point to an associated service routine. The vector is obtained from the SIO by the CPU generating an interrupt acknowledge cycle.

The interrupt method of communication between the SIO and the CPU is the one adopted within the Apricot. The SIO only generates an interrupt to the CPU when a channel requires a specific servicing routine to be carried out, (e.g. keyboard data available) thus leaving the CPU free to service the other devices on the board.

SIO Architecture

Internally, the SIO consists of four areas of circuitry:

1. A Processor interface.
2. Internal interrupt control logic.
3. The two independent serial communication channels, channel A and channel B.

The processor interface handles all communications via the data bus between the CPU and a series of internal registers, associated with each of the two serial channels. These are of three different types; data registers, command registers and status registers.

Both channels have two data registers each which are accessed by the CPU; one handles the transmit data from the 8086 - transmit buffer, and one stores receive data from the serial channel - receive buffer.

The mode and operation of each channel is determined by the contents of the command registers (Write Registers). These are initialised with control words prior to any data transfer over the two serial channels, and may be modified as data transfer operations proceed.

Seven Write Registers are associated with each channel. An eighth Write Register, which is located in channel B is shared by both channels.

The information contained in the status registers (Read Registers) indicate the status within each channel. Two Read Registers are associated with each channel. A third register accessed via channel B is common to both channels.

The interrupt control logic section of the SIO assigns the priority to the various sources of interrupts, generates the interrupt output to the CPU and produces the interrupt vector.

The priority of the interrupts is fixed with channel A always assigned a higher priority than channel B. The order of priority assigned within each channel is receiver interrupts (highest priority), followed by transmitter interrupts, followed by external/status interrupts (lowest priority).

Separate transmit and receive data paths are provided within the two serial communications channels.

Receive Path

The receiver ports are quadruply buffered by an input shift register and three storage registers. The shift register converts the incoming receive serial data into a parallel byte.

The three storage registers are configured in a FIFO (first in first out) arrangement. The first parallel data byte from the shift register is loaded into the bottom of the FIFO stack and then shifted through to the top at a rate, determined by an internal clock.

The register at the top of the stack is the receive buffer which acts as the storage buffer for the receive character until read by the processors. Reading the character in the receive buffer automatically transfers the next character (if any) to the top of the stack.

Error flags associated with each receive character are also buffered by a similar arrangement. These are loaded into the register of a parallel receive error FIFO stack at the same time as the character. Each time the receive character is shifted through the character FIFO stack, the error flags are moved accordingly. The top of the receive error FIFO stack is one of the Read Registers.

The contents of this Read Register reflect the status of the character stored in the receive buffer, but may also contain any receiver overrun and parity errors received from previous characters. These two error conditions remain within the status register, even when new character error flags are loaded, until cleared by an error reset command.

If the character FIFO stack is full (i.e. contains three characters) and the CPU fails to read the character within the receive buffer before another receive character is supplied to the stack, the last character placed in the stack is overwritten with the new character, and the receiver overrun flag is recorded in the corresponding register in the receive error FIFO stack.

The first two characters in the stack are never overwritten, even if more characters are received; the last character is continuously overwritten.

Error conditions stored in the FIFO stack can be programmed to generate an interrupt to the CPU, on being loaded into the Read Register at the top of the stack.

In the asynchronous mode, using 8 bits per character, the serial receive data is stripped of the start, stop and parity bits, prior to being supplied to the serial shift register.

For character lengths of less than 8 bits, the receiver circuits automatically insert logic 1's in the most significant places to assemble a byte of data, if a parity bit is not included with the character. If a parity bit is included, the parity bit is also assembled with the bits of the character, with any remaining bits set to logic 1.

For example, a 5-bit receive character with a parity bit is assembled in the receive buffer in the following format:

1	1	P	D4	D3	D2	D1	D0

D = 5-bit character.
P = Parity bit.

In the byte oriented synchronous modes, Monosync and Bisync, receive data is not transferred to the receive FIFO register stack until character synchronisation is established.

Detecting a byte of receive data which matches a sync character stored in a Write Register establishes synchronisation in Monosync. Detecting two consecutive bytes of receive data which match the sync characters stored in two of the Write Registers establishes synchronisation in Bisync.

Searching for the character sync is achieved by programming the channel to operate in the first phase of the synchronous reception process, termed the hunt phase. The second phase is the actual reception of data, where the receive data is automatically transferred to the FIFO stack after detection of the character sync.

Also included in the receive path in synchronous modes, is a CRC error detection circuit which sets an error flag according to the result of CRC comparisons.

The receiver circuits in the bit oriented synchronous modes operate in a similar manner to the byte oriented modes, operating with two separate phases, a hunt phase and a receive phase. Receive data is not transferred to the receive FIFO register stack until the hunt phase is sucessfully completed.

This requires detecting an opening flag sequence corresponding to a flag pattern, stored in one of the Write Registers. Reception is terminated on detecting the closing flag at the end of the message (EOM). This prevents any further data being supplied to the FIFO register stack. The receive data is also supplied to the CRC error detection circuits.

Transmit Path

The transmitter sections of the serial channels consist of an 8-bit data register (the transmit buffer), supplied with character data from the CPU, and a 20-bit transmit shift register. The shift register converts the parallel data in the transmit buffer into serial format and also performs a variety of other functions depending on the mode of operation.

In the asynchronous mode, the data from the transmit buffer is automatically formatted with start, stop and parity bits within the shift register, prior to transmission.

In Monosync and Bisync, the shift register is loaded with the sync characters stored in the Write Registers, at the beginning of the message, and then data from the transmit buffer as transmission proceeds. The shift register also supplies the serial data to the CRC generator, which produces the two byte CRC at the end of the message.

In the bit oriented modes, the shift register is loaded with the flags stored in the Write Register, at the beginning and end of the message. The shift register supplies the serial data to the CRC generator which generates the 2-byte CRC at the end of the data field, and to the transmit data output.

For character lengths of less than 8 bits, the characters sent to the transmit buffer have to be right justified, by the programmer. The logic state of the unused bits within each character byte (the MSB) are immaterial for character lengths of 6 and 7 bits, since the extra bits are automatically ignored by the SIO.

For a 5-bit character, the unused bits have to be programmed with logic 0's. Character lengths of less than 5 bits require a combination of logic 1's and logic 0's, to be inserted in the MSB as detailed in a later section.

Processor Interface

The connections to the SIO processor interface are detailed in the table "System Connections" at the end of the chapter. The interface handles the transfer of data, commands and status information (via the system data bus), between the CPU and the series of addressable registers within the SIO.

The system software views the SIO as four bi-directional peripheral ports, located in the system I/O space. The port addresses, as defined by the SIO select and the system address bus connections, are detailed below.

Address	Port
20H	Channel A Tx/Rx data
22H	Channel A commands/status
24H	Channel B Tx/Rx data
26H	Channel B commands/status

Writing data to the I/O address location 20H loads data into the transmit buffer of channel A (sound data for transmission); reading data from the same location accesses data stored in the receive buffer of channel A (Keyboard/mouse data).

Writing data to the I/O address location 24H loads data into the transmit buffer of channel B (RS232 transmit data); reading data from I/O address location 24H accesses data stored in the receive buffer of channel B (RS232 receive data).

Writing data to the Write Registers (command registers) and reading data from the Read Registers (status registers) in the two channels, generally requires two data transfers.

The first transfer is a write to the commands/status address of the channel, and provides the SIO with a pointer to determine the register for the next read or write operation.

If the next operation is writing to the command/status address location, a command byte is loaded into the Write Register specified by the pointer. If the next operation is reading from the commands/status location, a status byte is accessed from the Read Register specified by the pointer.

Following the second transfer operation the pointer within the SIO is always cleared to zero.

Write Register Definition

Each channel contains seven Write Registers, which configure the SIO to match the desired mode and application. Five of these registers are command registers which define the basic mode of operation (e.g. asynchronous), and configuration of the channel within the mode (e.g. number of bits per character, etc.).

The two other registers are sync character registers. An eighth register, which is written to via channel B, is common to both channels and is programmed with the base address of the interrupt vector.

A summary of the functions of the Write Registers is provided in tabular format below. This is followed by a more detailed description of the commands for each individual register.

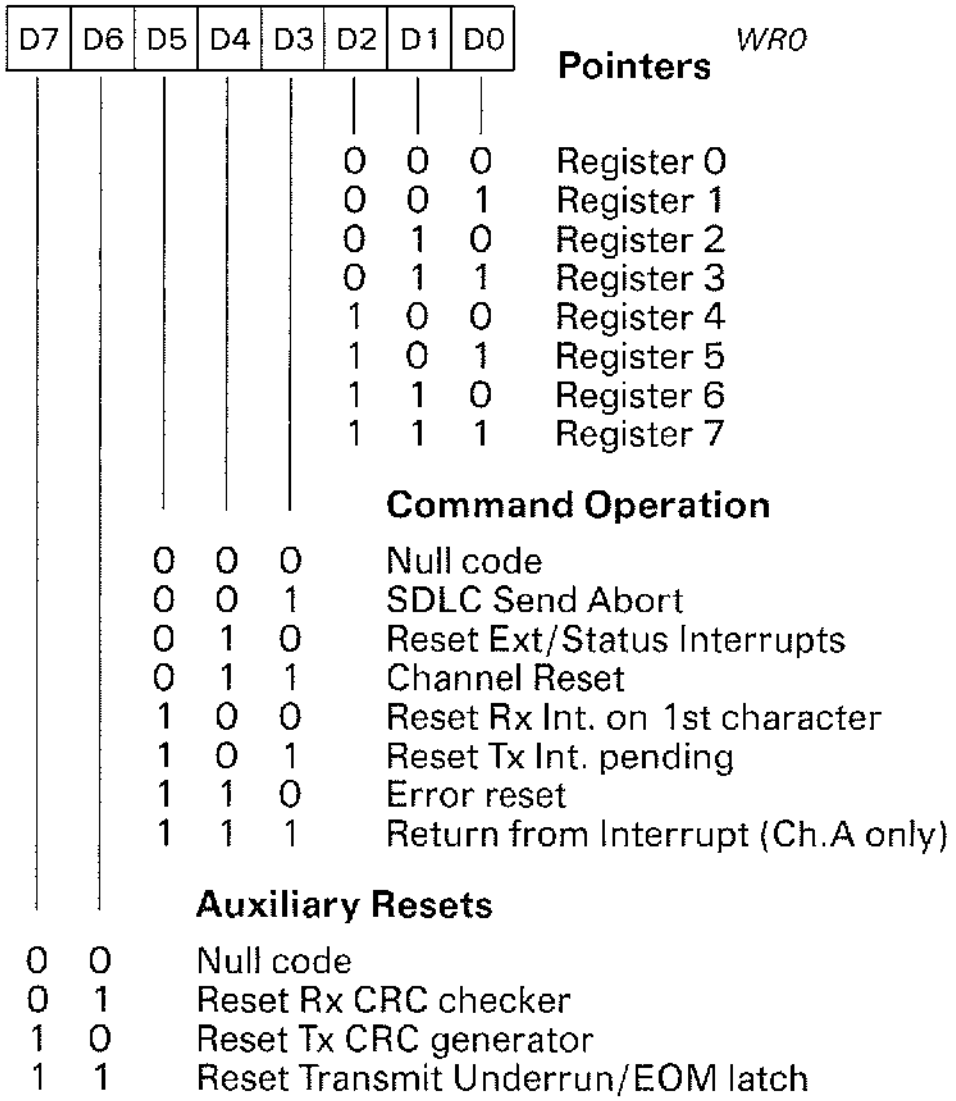
The only constraint on the order of programming the registers is that initialising each channel of the SIO following a hardware system reset or a software channel reset, requires the parameters of Write Register 4 to be issued before the parameters/commands of Write Registers 1, 3, 5, (6 and 7 if used).

As a general rule, if changing from one operational configuration to another (e.g. 8-bit asynchronous to 7-bit asynchronous), the whole initialisation sequence should be repeated with the new parameters.

Write Register Summary

WR0	Write Register 0. Allows the programmer to perform a number of basic reset functions in addition to acting as the pointer to select the register for the next command/status transfer operation.
WR1	Write Register 1. Contains the bits which select the interrupt mode, and allows the interrupts to be enabled/masked.
WR2	Write Register 2. This register is programmed with the base address of an interrupt vector and is addressed through channel B only.
WR3	Write Register 3. The bits written to this register configure and control the channel receiver circuitry.
WR4	Write Register 4. This register is programmed with bits which select the mode of operation (asynchronous or synchronous), and some of the operational parameters of the selected mode.
WR5	Write Register 5. The bits written to this register configure and control the channel transmit circuitry.
WR6	Write Register 6. This register is programmed with a sync character in Monosync and Bisync, and a check address in the bit oriented synchronous modes.
WR7	Write Register 7. This register is programmed with a sync character in Monosync and Bisync, and a flag character in the bit oriented synchronous modes.

Write Register 0


D7 D6 D5 D4 D3 D2 D1 D0
WR0
Pointers
0 0 0 Register 0
0 0 1 Register 1
0 1 0 Register 2
0 1 1 Register 3
1 0 0 Register 4
1 0 1 Register 5
1 1 0 Register 6
1 1 1 Register 7
Command Operation
0 0 0 Null code
0 0 1 SDLC Send Abort
0 1 0 Reset Ext/Status Interrupts
0 1 1 Channel Reset
1 0 0 Reset Rx Int. on 1st character
1 0 1 Reset Tx Int. pending
1 1 0 Error reset
1 1 1 Return from Interrupt (Ch.A only)
Auxiliary Resets
0 0 Null code
0 1 Reset Rx CRC checker
1 0 Reset Tx CRC generator
1 1 Reset Transmit Underrun/EOM latch

Pointers (D0 to D2)

These bits signify the register for the next command/status transfer operation. If the next operation is a write, the pointer specifies a write register. If the next operation is a read, the pointer specifies a read register. Following a read or write to any register (except WR0), the pointer points to register 0 (i.e. Write Register 0 or Read Register 0).

Commands (D3 to D5)

These bits specify eight different commands, the function of which are detailed below.

1. Null Code. This command enables the programmer to specify the next register for a command/status transfer using the pointer bits, without affecting the operation of the SIO.
2. SDLC Send Abort. Used only in the bit oriented synchronous modes. The command causes the SIO to generate an abort sequence which informs the receiver of data from the SIO to terminate reception.
3. Reset Ext/Status Interrupts. After an External interrupt (caused by a change of state on a modem control input), or a Status interrupt (caused by detecting a break/abort condition in receive data or a transmit underrun/EOM condition in transmit data), corresponding status bits within Read Register 0 are latched. This command re-enables the bits and allows further interrupts to occur.
4. Channel Reset. This command performs the same function as a hardware reset but on the specified channel only. Issuing the command to channel A also resets the interrupt logic, clearing any current and all pending interrupts. All Write Registers in the channel must be reprogrammed, following the command.
5. Reset Rx Interrupt on First Character. If the Interrupt on First Receive Character Mode is in operation (bits 3 and 4 of WR1), an interrupt is generated on receiving the first character. At the end of the message, this command is issued to reactivate the mode.
6. Reset Tx Interrupt Pending. If the Transmit Interrupt is enabled (bit 1 of WR1), the SIO generates an interrupt every time the transmit buffer is empty. Issuing the Reset Tx Interrupt Pending command, resets the transmit interrupt, and prevents the interrupt being raised again until the transmit buffer is loaded with a character and becomes empty again.
7. Error Reset. Parity and Overrun errors are latched into Read Register 1 (the status register at the top of the receive error FIFO stack). These error conditions remain within the register until the Error Reset Command is issued.

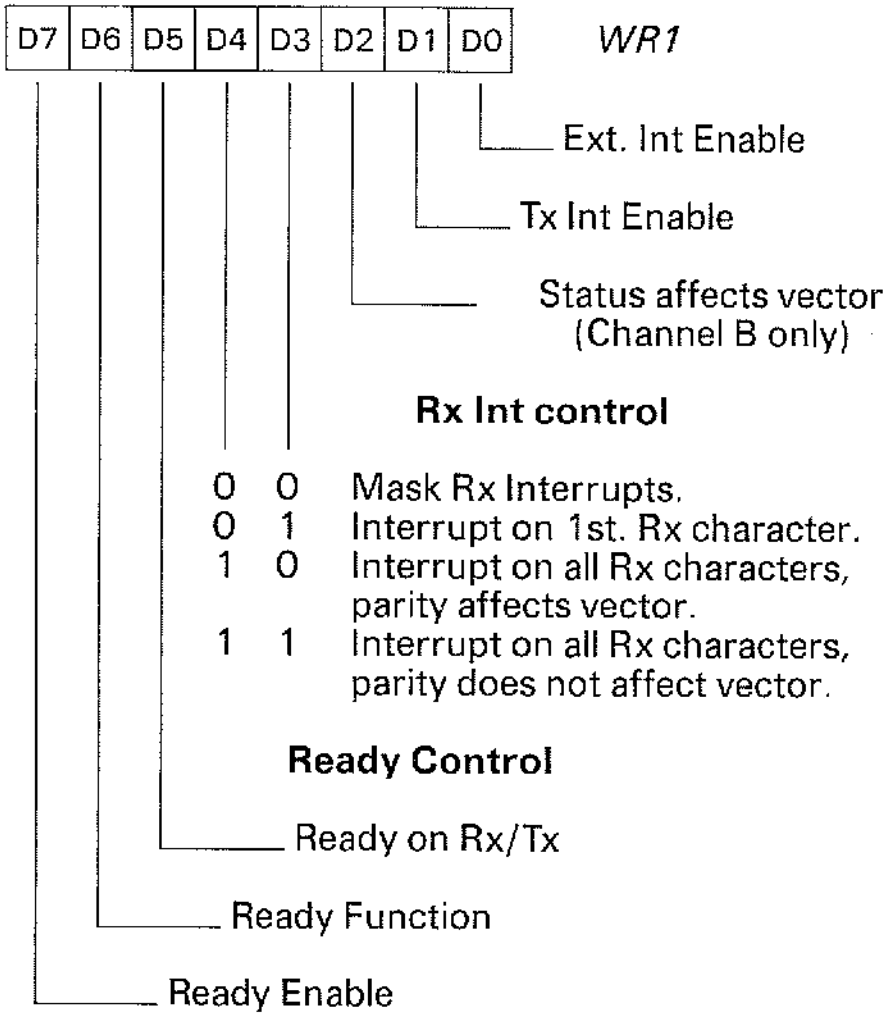
8. Return from Interrupt. This command is used to signify to the SIO that the interrupt routine in service specified by the last interrupt vector read by the CPU has been completed. The command is issued through channel A only and resets the in-service interrupt within the SIO, and allows the highest priority interrupt pending (if any) to interrupt the CPU.

Auxiliary Resets (D6, D7)

These bits specify four commands, the function of which are as described below.

1. Null Code performs the same function as previously described for the Null Code of the Command bits.
2. Reset Rx CRC Checker. This command resets the CRC error detection circuitry located in the channel receiver circuit.
3. Reset Tx CRC Generator. This command resets the CRC generator located in the channel transmitter circuit.
4. Reset Tx Underrun/EOM. When the SIO detects either a Transmit Underrun condition or the End of a Message (EOM), a corresponding bit is set within Read Register 0 (bit 6). The bit is reset by issuing this command.

Write Register 1


D7 D6 D5 D4 D3 D2 D1 D0
WR1
Ext. Int Enable
Tx Int Enable
Status affects vector
(Channel B only)
Rx Int control
0 0 Mask Rx Interrupts.
0 1 Interrupt on 1st. Rx character.
1 0 Interrupt on all Rx characters, parity affects vector.
1 1 Interrupt on all Rx characters, parity does not affect vector.
Ready Control
Ready on Rx/Tx
Ready Function
Ready Enable

Ext. Int Enable (D0)

The logic state of this bit determines whether the External/Status interrupts are enabled (logic high) or masked (logic low). If enabled, an interrupt is produced as a result of the following conditions.

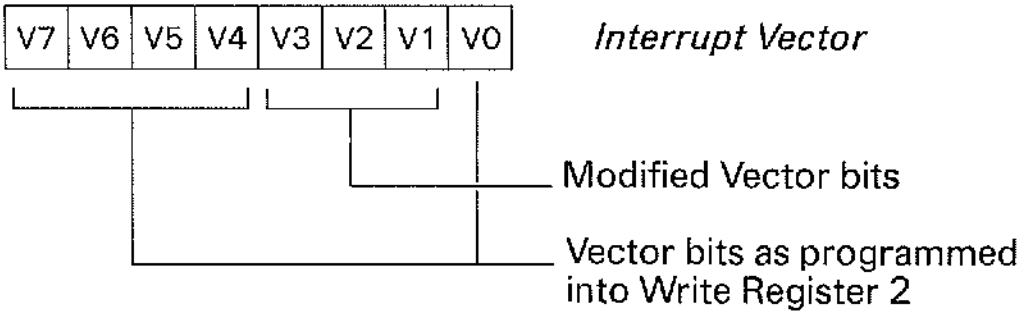
1. Transitions on the modem control lines; $\overline{\text{DCD}}$, $\overline{\text{CTS}}$.
2. Detecting a break or abort condition in the receive data.
3. At the start of transmission of either CRC or sync characters, in synchronous modes, when the Transmit Underrun/EOM latch becomes set.

Tx Int Enable (D1)

The logic state on this bit determines whether the transmit interrupt is enabled (logic high) or masked (logic low). If enabled, an interrrupt is produced when the transmit buffer becomes empty.

Status Affects Vector (D2)

This bit is active only in channel B. If set to logic low, the interrupt vector programmed in Write Register 2 is written into the Read Register 2 in channel B unmodified. If set to logic high, the interrupt vector is modified according to the interrupting condition as detailed below.



Modified Vector Bits

V3	V2	V1	Interrupt Condition
0	0	0	Ch.B Transmit buffer empty
0	0	1	Ch.B Ext/status Interrupt
0	1	0	Ch.B Receive Character Available
0	1	1	Ch.B Special Receive Condition*
1	0	0	Ch.A Transmit buffer empty
1	0	1	Ch.A Ext/status Interrupt
1	1	0	Ch.A Receive Character Available
1	1	1	Ch.A Special Receive Condition*

* Special Receive Conditions include Parity error, Rx Overrun Error, Framing Error, End of Frame (SDLC).

Rx Int Control bits (D3, D4)

These two bits select the various receive conditions able to generate an interrupt.

1. Mask Rx Interrupts. Disables all receive interrupts.
2. Interrupt on 1st Rx Character. Enables an interrupt to be generated on detecting:

   (a) The first receive character in the receive buffer.
   (b) Any special receive condition.

3. Interrupt on all Rx Characters, parity affects vector. Enables an interrupt to be generated on detecting:

   (a) Any receive character within the receive buffer.
   (b) Any special receive condition.

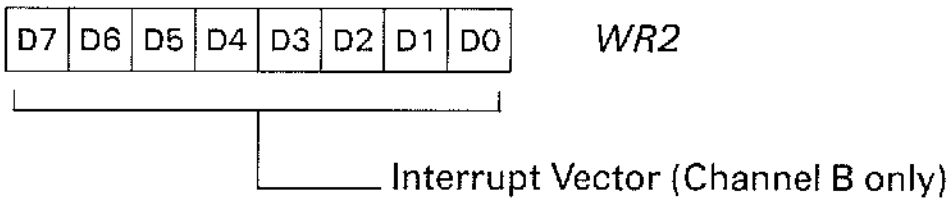
4. Interrupt on all Rx Characters, parity does not affect vector. Enables an interrupt to be generated on detecting:

   (a) Any receive character within the receive buffer.
   (b) Any special receive condition apart from a Parity error.

Ready controls (D5 to D7)

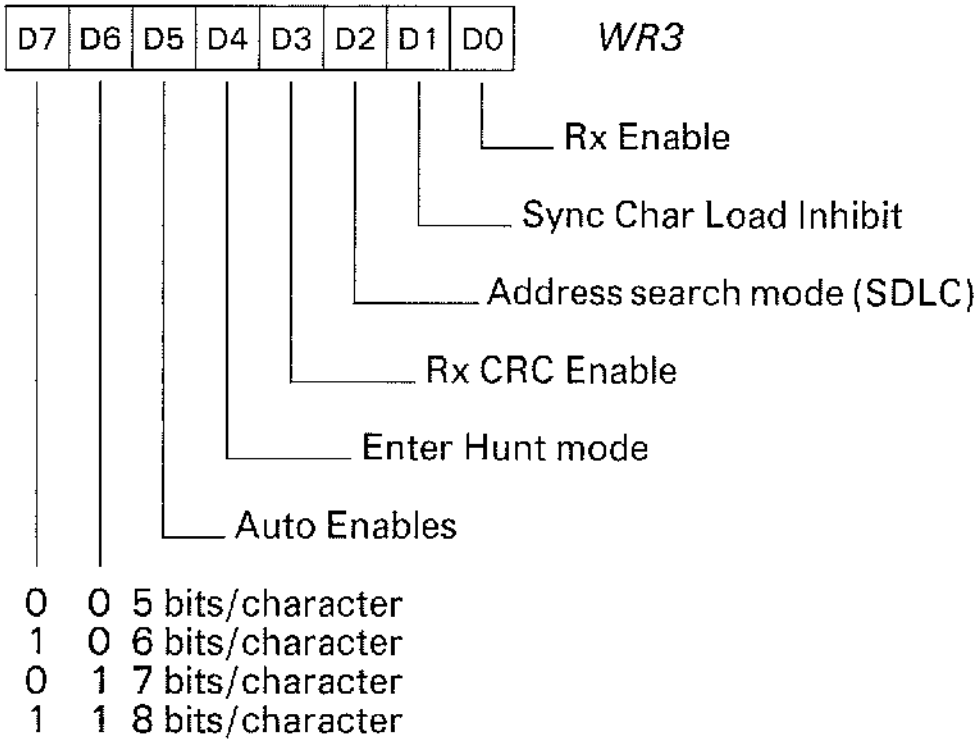
These three bits control the state of the $\overline{W/RDY}$ outputs from the SIO and are not connected in the F1. The state of the three bits is thus immaterial.

Write Register 2



Write Register 2 is programmed with the base address for the interrupt vector (50H) read from Read Register 2 during the interrupt cycle. If the Status Affects Vector bit within Write Register 1 (WR1, D2) is set to logic high, the interrupt vector read by the programmer is the base address with the LSB modified as previously described. If the Status Affects Vector bit is set to logic low, the interrupt vector in the Read Register corresponds to the base address unmodified.

Write Register 3



Rx Enable (D0)

If set to logic high, the receiver circuits are enabled to accept data. If set to logic low, the receiver circuits are disabled.

Sync Character Load Inhibit (D1)

Setting this bit to logic high prevents any sync characters being loaded into the receive FIFO stack.

Address Search Mode (D2)

If a bit oriented synchronous mode is selected and this bit is set to logic high, only messages with an address (following the opening flag) matching either the programmed address in Write Register 6 or the global address (FFH), are accepted by the receiver circuits. If set to logic low, no address search is carried out.

Rx CRC Enable (D3)

This bit enables (logic high)/disables (logic low) the receiver CRC error detection circuits.

Enter Hunt Mode (D4)

The Hunt mode in synchronous modes is automatically entered following a reset. If character synchronisation is lost (any Sync mode) or the incoming data is not required (SDLC), the hunt phase can be re-entered by setting D4 to logic high.

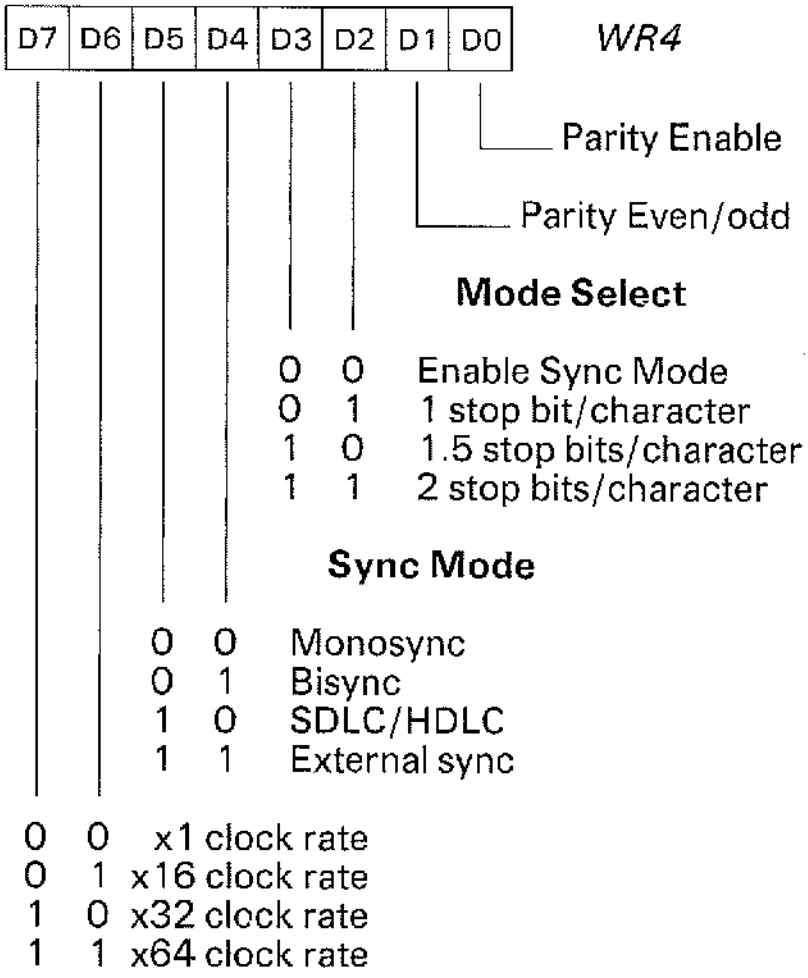
Auto Enables (D5)

Setting this bit to logic high allows the two modem input control lines, DCD and CTS to control data transfers over the serial data link. DCD controls the SIO receiver circuits, CTS the transmitter circuits.

Rx bits character (D6, D7)

The combination of these two bits determine the number of receive bits assembled to form a character.

Write Register 4



Parity Enable (D0)

If this bit is set to logic high, a parity bit is added to the transmit character data and is expected in receive data.

Parity Even/Odd (D1)

This bit is used when the Parity Enable bit is set to logic high to determine the sense of the parity code in the transmit data and the expected parity code in receive data. Even parity is signified by setting this bit to logic high, odd parity by setting the bit to logic low.

Mode Select (D2, D3)

The combination of these two bits differentiate between asynchronous and synchronous modes of operation and also specify the number of stop bits added to the transmit data in the asynchronous mode. The receiver circuits always check for one stop bit in the receive data, regardless of the number of stop bits in the transmit data.

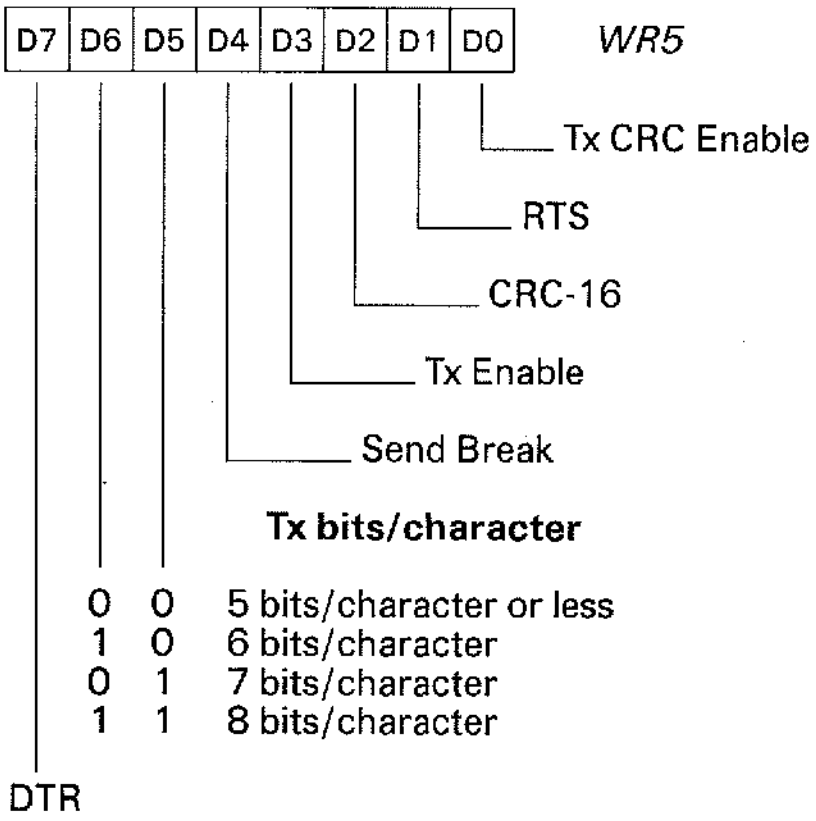
Sync Mode (D4, D5)

If synchronous mode is selected by the Mode Select bits, these bits determine the type of synchronous mode, for transmit and receive data.

Clock Rate (D6, D7)

These bits specify the multiplier applied to both the transmit and receiver input clock rate prior to being used to set the transmit and receive baud rates. For synchronous modes the x1 clock rate must be selected. Any rate may be selected for the asynchronous mode, apart from the x1 rate.

Write Register 5



Tx CRC Enable (D0)

This bit enables (logic high)/disables (logic low) the CRC generator in the transmit data path.

RTS (D1)

This bit controls the state of the modem control output Request to Send (RTS). When the RTS bit is set to logic high, the RTS output is set active (logic low/line spacing). When the RTS bit is reset to logic low, the RTS output is reset to the inactive logic high state (line marking).

In asynchronous mode, the RTS output is not reset by setting the RTS bit low, until after transmission of the character in the transmit buffer.

CRC-16 (D2)

This bit selects the CRC polynomial used by the transmitter and receiver circuits. If set to logic high, the CRC-16 polynomial ($X^{16} + X^{12} + X^5 + 1$) is selected. If set to logic low, the SDLC polynomial ($X^{16} + X^{15} + X^2 + 1$) is selected.

Tx Enable (D3)

This bit acts as the enable/disable control signal for the transmitter output. Character data in the transmit buffer cannot be transmitted or a break condition sent unless this bit is set to logic high. If set to logic low, the transmit output is held in the line idle marking condition.

Send Break (D4)

When set to logic high and the transmitter is enabled, the transmit data output is forced to the spacing condition, regardless of any data transmission in progress.

Tx bits/character (D5, D6)

The combination of these two bits specify the number of bits per character in the transmit data. Characters have to be assembled into the correct format by the programmer. All character lengths of less than 8 bits have to be right justified, with the following format.

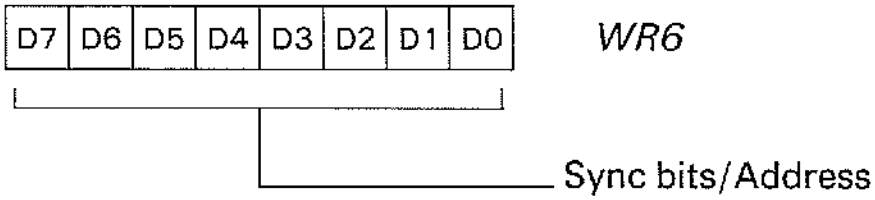
D7	D6	D5	D4	D3	D2	D1	D0	Bits/Character
0	D	D	D	D	D	D	D	7
0	0	D	D	D	D	D	D	6
0	0	0	D	D	D	D	D	5
1	0	0	0	D	D	D	D	4
1	1	0	0	0	D	D	D	3
1	1	1	0	0	0	D	D	2
1	1	1	1	0	0	0	D	1

D = Character data bits

DTR (D7)

This bit controls the modem control output Data Terminal Ready (DTR). When the DTR bit is set to logic high, the DTR output is set to the active state (logic low). When reset to logic low, the DTR output is reset to the inactive logic high state.

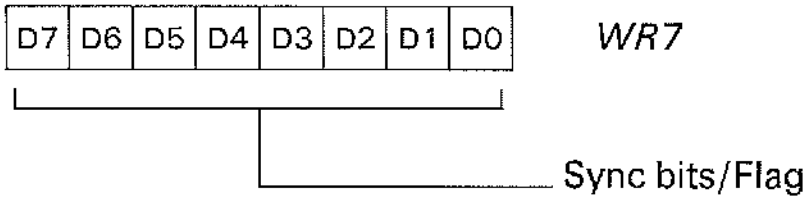
Write Register 6



Write Register 6 is programmed with:

1. The transmit sync character in Monosync.
2. The 8 LSB of the 16-bit sync character in Bisync.
3. The 8-bit frame address byte in bit oriented modes.

Write Register 7



Write Register 7 is programmed with:

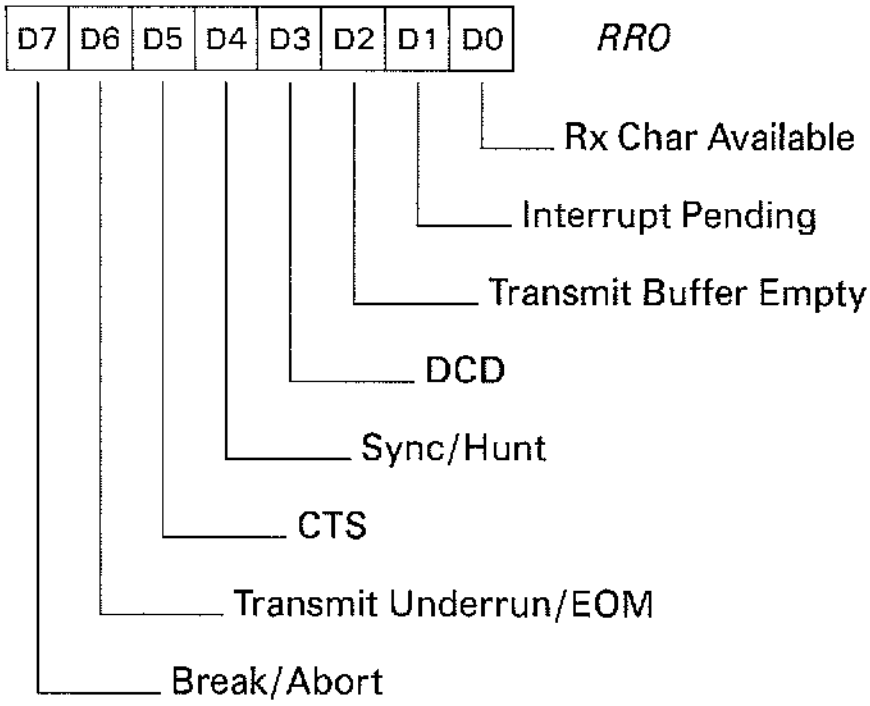
1. The receive sync character in Monosync.
2. The 8 MSB of the 16-bit sync character in Bisync.
3. A flag character in bit oriented modes.

Read Register Definition

Both channels contain two Read Registers each (RR0 and RR1), which define the status within the channel. A third Read Register (RR2), accessed through channel B, contains a copy of the interrupt vector which indicates the SIO interrupt routine (if any) currently in service.

Reading the contents of Read Registers 1 and 2 requires a two byte transfer operation. The first, a write operation to the commands/status address location using the pointer bits of Write Register 0 to specify the Read Register: the second, the actual read operation from the same address to access the contents of the register. Following any read or write operation (apart from writing to Write Register 0), the address pointer within the SIO is always cleared to zero, allowing the contents of Read Register 0 to be accessed using a single read operation.

Read Register 0



Rx Char Available (D0)

This bit is held at logic high until all characters within the receive FIFO stack are read. Logic low indicates that no more receive characters are available.

Interrupt Pending (D1)

This bit only has significance in channel A. In channel B, the bit is always at logic low. When set to logic high, the bit indicates that an interrupt condition is present within the SIO.

Transmit Buffer Empty (D2)

This bit is set to logic high every time a character is transmitted out of the transmit buffer. It is reset to the logic low inactive state by reloading the transmit buffer with a new data character.

DCD (D3)

The DCD bit indicates the state of the modem control input Data Set Ready (DSR) in channel B and the state of the modem control input DCD in Channel A. The state of the bit is latched every time any external/status interrupt condition occurs, and remains in the latched state until reset by writing the reset external/status interrupt command to Write Register 0.

Therefore, to ensure that the current state of the DCD/DSR input is obtained, the bit should be read immediately following a reset external/status interrupt command. The DCD bit indicates the inverse of the state on the DCD/DSR input.

Sync/Hunt (D4)

This bit only has any significance in channel B. The bit reflects the phase of the synchronous receive operation.

Initiating the Hunt phase causes the bit to be set to logic high. (i.e. On reset or setting the Enter Hunt Mode bit in Write Register 3).

On achieving character synchronisation, the bit is set to logic low as the receive phase begins and remains in this condition unless the Hunt phase is initialised again.

CTS (D5)

This bit functions in a similar manner with regard to the latching process, but indicates the inverse of the state on the Clear to Send input (CTS).

Transmit Underrun/EOM (D6)

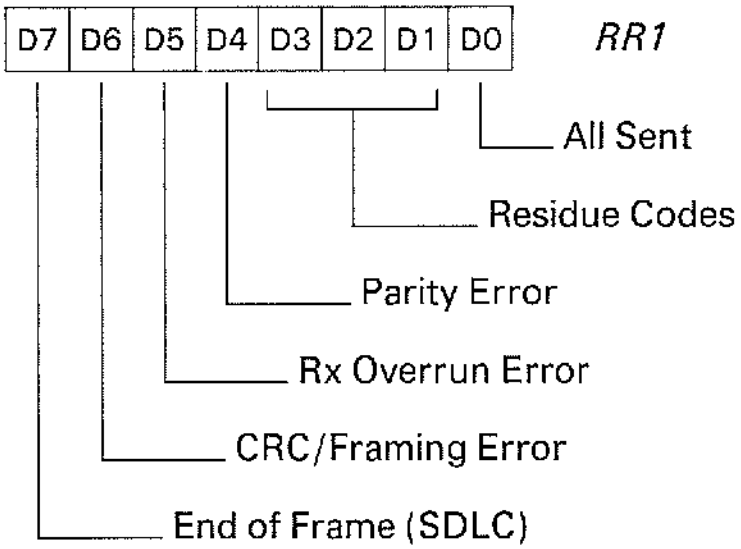
In synchronous modes, the bit is set to logic high following a system/channel reset, allowing sync/flag characters to be sent when the transmit buffer becomes empty. When the reset transmit underrun/EOM command is issued to Write Register 0, the transmit underrun/EOM bit is set to logic low. This enables CRC characters to be automatically sent instead of the sync/flag characters.

Break/Abort (D7)

In asynchronous modes, this bit is set to logic high, when a break is detected in the receive data. The bit is not reset until, a reset external/status command is issued to Write Register 0 and the break condition is removed.

In the bit oriented modes, the bit is set to logic high on detecting an abort sequence in the receive data. The bit is reset to logic low on loading Write Register 0 with the reset external/status command. The bit is not used in the byte oriented synchronous modes.

Read Register 1



All Sent (D0)

In asynchronous modes, this bit is set to logic high when all the bits of the character have been transmitted onto the serial link.

In synchronous modes the bit is permanently set to logic high.

Residue Codes (D1 to D3)

The combination of these three bits indicate the length of the I-field in the bit oriented modes where the I-field is not an integral multiple of the character length.

Parity Error (D4)

When parity is enabled, this bit is set to logic high on detecting a receive character whose parity does not match the sense programmed by bit 1 of Write Register 4. The bit remains set in the error condition until reset by loading the error reset command into Write Register 0.

Rx Overrun Error (D5)

This bit is set to logic high when one or more receive characters have been overwritten in the receive FIFO buffer. The bit remains set in the error condition until reset by loading the error reset command into Write Register 0.

CRC/Framing Error (D6)

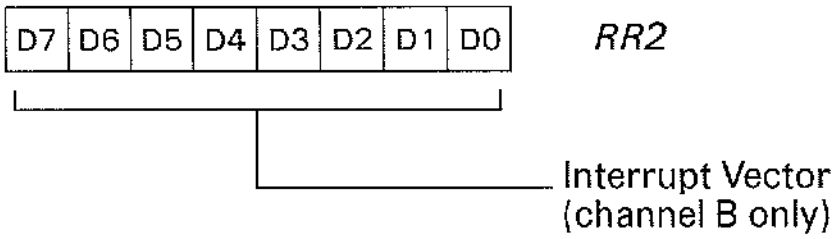
The function of this bit is dependent on the mode selected. In asynchronous modes, the bit is set to logic high on detecting a receive character with incorrect stop bits (framing error). The error condition only persists for the particular character stored in the receive buffer.

In synchronous modes, the bit indicates the result from the receive CRC error detection circuit. A logic high indicates a CRC error. The error conditions are reset to the inactive low state after issuing the error reset command to Write Register 0.

End of Frame (D7)

In the bit oriented modes, this bit indicates that a valid closing flag has been detected and the CRC Error and residue codes are now valid. The bit is reset by issuing an error reset command to WR0.

Read Register 2



Read Register 2 contains the interrupt vector and is read through channel B only. If the status affect vector bit is set (D2, Write Register 1), the register indicates the current interrupt service routine (if any) in operation.

If no interrupts are pending, the vector is set to the condition for a special receive condition in channel B (see Write Register 1 description). If the status affect vector bit is not set (logic low), the register contains a copy of the vector written into Write Register 2 in channel B.

SIO Interrupt Sequence

The SIO incorporates an elaborate interrupt structure, which acts in conjunction with the interrupt structure provided by the Z80 CTC. This is fully described in the chapter "Interrupt Control". The following paragraphs merely detail the interrupt structure of the SIO.

A single interrupt output line ($\overline{INT}$) connects the SIO to the CPU.

Both channels within the SIO are able to generate an interrupt for a variety of conditions. All interrupting sources can be enabled or disabled (masked) by software, and are ordered on a priority basis.

All sources within Channel A are assigned a higher priority than Channel B. Within each channel, the assigned priority is as detailed below:

1. Special receive condition (Highest).
2. Receive character available.
3. External/status interrupt.
4. Transmit data required (Lowest).

If a SIO interrupt is in service when a higher priority interrupt condition occurs, the higher priority condition is granted service. The SIO stores the lower priority condition and resumes the interrupt service routine on completion of the higher priority interrupt routine.

A special receive interrupt is generated when the SIO detects any of the following conditions in the receive data:

1. Parity errors.
2. Receiver overrun.
3. Framing errors (asynchronous modes).
4. End of Frame (SDLC/HDLC only).

Receive character available interrupts can be programmed to generate an interrupt on every receive character or for the first character only (synchronous modes).

An external/status interrupt can be caused by any of the following conditions:

1. Transitions on the modem control lines; $\overline{DCD}$, $\overline{CTS}$.
2. A break in receive data (asynchronous modes).
3. An abort sequence in receive data (bit synchronous modes).
4. Transmit underrun/EOM condition in transmit data (synchronous modes).

The actual sequence of events performed on the occurrence of a SIO interrupt condition is as follows.

The SIO compares the new interrupt with any interrupt currently in service. If the new interrupt is of a higher priority than the current interrupt, the $\overline{INT}$ output is set active (logic low). If the new interrupt is of a lower priority, the interrupt condition is stored until it becomes the highest priority.

The CPU determines the process generating the interrupt by performing an interrupt acknowledge cycle which provides an associated interrupt vector, which specifies the source of interrupt.

This interrupt vector is constructed from a base address previously supplied to the SIO (50H) modified according to the condition generating the interrupt.

The SIO interrupt thus allows the programmer to select the appropriate interrupt service routine. On completion of the SIO service routine, the CPU has to issue a return from interrupt command (to WR0), to enable any lesser priority pending interrupts to generate an active interrupt output.

The SIO is able to produce eight different interrupt vectors according to the type of interrupt as detailed on the previous pages (4 vectors for each channel). In the case where the interrupt can be caused by a number of different conditions (i.e. special receive and external/status), the actual cause can be detected by reading bits within the appropriate status register.

Keyboard/Mouse Data

Transfer of data from the keyboard/mouse is handled by transmit channel A of the SIO. The channel is programmed to support synchronous communications at a fixed data rate (approximately 3.85 kbits per second).

The format of the data transmitted from both the keyboard and mouse is identical, both consisting of a 4 byte synchronous packet. The first byte in the packet is the sync byte (5AH). The remaining three contain data. (This is true in all cases apart from when the Keyboard System Reset button is pressed. In this case, contiguous sync bytes are transmitted to the Systems Unit instead. The function of this is to generate a hardware reset. This mechanism is discussed in the System Detail chapter).

The clock for the synchronous data is inherent in the data stream transmitted to the Systems Unit. This is split into separate Monosync data and clock waveforms by a signal conditioning circuit.

Reception of the incoming synchronous data consists of two separate phases. The first phase is the Hunt phase, where channel A of the SIO analyses all incoming data, searching for the sync byte 5AH. This is the header byte for all valid keyboard and mouse transmissions.

Detecting the sync byte automatically switches the SIO into the second phase, the Receive phase, where the three bytes of data following the header are loaded into the three-byte buffer of Channel A.

Every time the SIO tranfers received data into the top register of the three-byte buffer, it generates an interrupt to the CPU.

The Systems Unit does not exercise control of data flow over the IR link (all transfer of information is one-way only). To prevent any loss of data, the CPU has to read the incoming data at a fast enough rate before it is overwritten in the receive FIFO stack.

The software must therefore be capable of processing data stored in the Channel A SIO receive data buffer at the maximum possible incoming receive data rate. This corresponds to the minimum time taken between the end of one packet and the beginning of the next consecutive four byte packet from the Keyboard and is of the order of 20 ms.

Encoding the data into a synchronous packet format provides a high degree of protection to interference from stray infra-red transmissions generated by other sources. It prevents the system being unnecessarily interrupted by other sources, since only transmissions which contain a sync byte of 5AH will be considered to be valid.

Further protection to sources of interference is provided in the coding of the bytes following the sync header. These three bytes are encoded using a Hamming format. This is totally transparent to the SIO. The SIO is only concerned with receiving bytes of data and is not concerned with the make-up of the bytes. All "de-hamming" is carried out by the BIOS.

A description of the format, character codes and their significance, supplied from the Keyboard is detailed in the Keyboard chapter. Details of the Mouse and its transmission format is packaged into a separate manual.

Sound Generation

Sound can be generated by one of two methods.

The first method can be used to produce audio tones and simple noises.

The second method can be used to produce much more complex waveforms and synthesised sounds.

The first method requires programming channel A of the SIO to repeatedly transmit a sync byte to the audio amplifier. This is the sync header byte of a normal monosync transmission. The regularity of the bit pattern within the 8-bit sync byte controls the purity of the output tone. The frequency of the output is controlled by timer pulses generated by Channel 2 of the CTC. These pulses act as the baud rate control for the synchronous transmission.

The second method requires programming channel A of the SIO to transmit a series of data bytes, as in a standard monosync transmission. Complex sounds can be produced by varying the data sent to the SIO in combination with varying the frequency of the output via control of the CTC.

The two methods require slightly different programming sequences. These are described in the chapter headed "Sound Generation".

Channel A Programming Details

Channel A provides a series of Write Registers to configure the SIO to interface with the keyboard and mouse, and also to generate sound.

Copy Registers

As the write registers are write-only and cannot be read, the BIOS always keeps a copy of certain SIO registers which may be of use to programmers. These are the registers WR1, WR3, WR4 and WR5 in channel B.

The register copies are located in RAM in a register copy table along with copies of other write only registers used in the system. The reason for keeping copies is to avoid the possibility of adverse effects arising when the BIOS and the application programmer both update bits within the register. Since either source could unknowingly modify other bits within a particular register which are vital for the other's operation, a programming structure has been devised to minimise this problem.

The BIOS updates the write register by performing the following sequence:

1. Reads in the data byte stored in the copy register.
2. Modifies the bits within the byte as appropriate, leaving all other bits untouched.
3. Writes the modified byte to the Write Register.
4. Updates the copy register.

To avoid unintentionally overwriting bits set by BIOS, the same approach should also be adopted by the application programmer when directly accessing the hardware.

A copy of a particular register is located by:

1. Reading a double word pointer located at absolute address location 722H (the start of the register copy table).
2. Adding an offset to the pointer to form the address of the register copy byte of interest.

The offset for the Channel A copy registers are as follows:

Write Register 1	04H
Write Register 3	05H
Write Register 4	06H
Write Register 5	07H

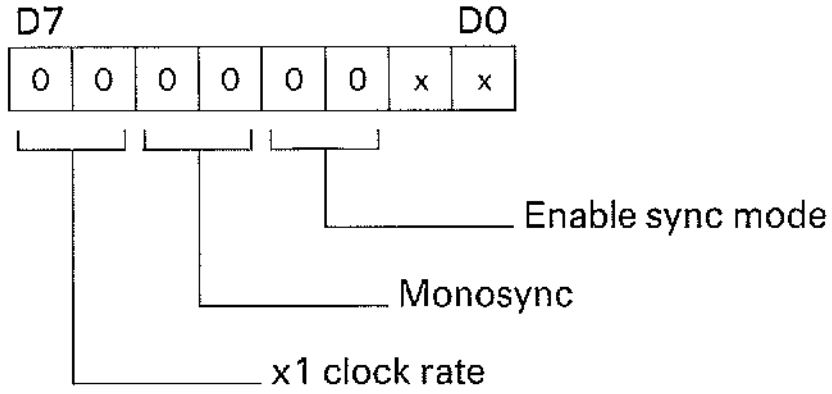
Initialisation

The parameters issued to the Write Registers of channel A for interfacing the SIO to the Keyboard are programmed during an initialisation routine by the BIOS.

Once channel A is initialised with the keyboard parameters and channel B is programmed with the correct base vector and the Status affects vector bit is set (Channel B - Write Register 1 bit D2), transfer of data between the Keyboard and the CPU can proceed via the synchronous hunt/receive mechanism.

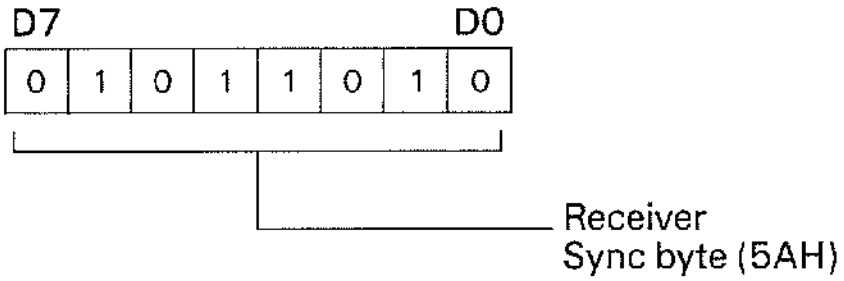
The format of the command bytes supplied to the Write Registers of channel A during initialisation are detailed in the following pages. The address locations utilised by channel A are described in a previous section, under the heading "Processor Interface".

Write Register 4

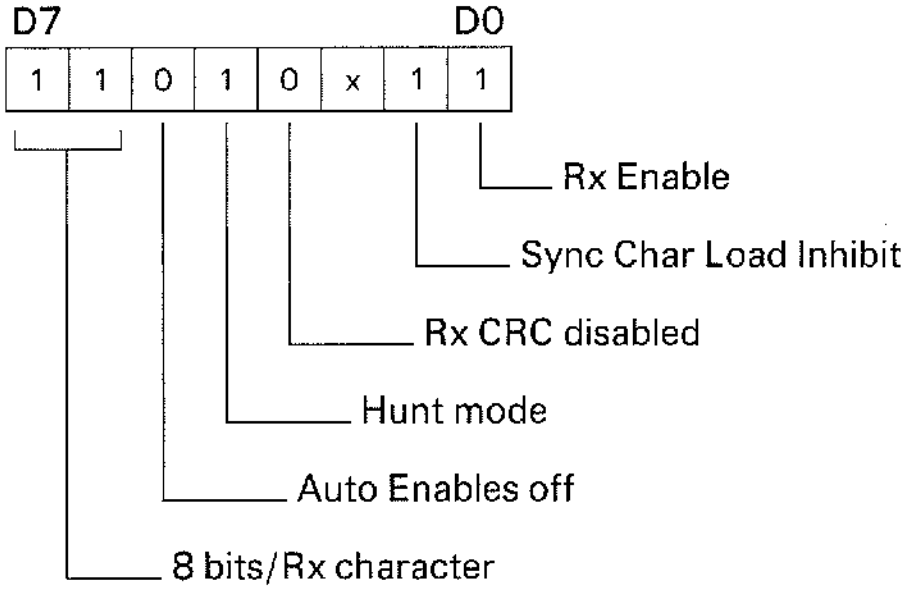


x = not applicable to mode and application, program to 0.

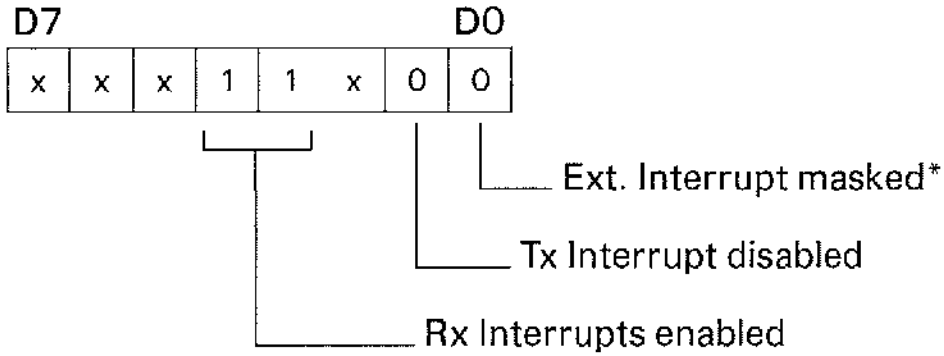
Write Register 7



Write Register 3



Write Register 1



* unmasked to allow the RS232C line $\overline{\text{DCD}}$ to generate an interrupt

Generating Sound

Once the channel has been initialised with the necessary parameters to receive data from the keyboard, it can then be programmed for generating sound. The registers used for programming the SIO to generate sound are detailed in the Sound Generation chapter.

RS232C Communications

General

The interface to the RS232C serial link is implemented by channel B of the SIO, which can be programmed to operate in either asynchronous or synchronous communications modes with transmit and receive baud rates determined either via the Z80 CTC, or via the external data communications equipment.

The clock inputs to the SIO pins TxCB and RxCB, are used to determine the transmit and receive baud rates for the RS232C link. Split rate clocks cannot be set up by the internal timer. The same rate is used for both transmit and receive.

Details of the timer and the appropriate programming values to produce the majority of the commonly used baud rates are detailed in the "Timer" chapter. These values apply to asynchronous communications only and require channel B to be programmed in the x16 clock rate mode to achieve the correct baud rate.

The selected transmit rate clock (TxCA) is also supplied to the RS232C connector (connector pin, TxClk), for use by the external data communications equipment.

Four of the most commonly used RS232C modem control lines are connected to various inputs and outputs of channel B of the SIO, available for coordinating data transfers over the serial link. These include the outputs $\overline{\text{RTS}}$ and $\overline{\text{DTR}}$, and the inputs $\overline{\text{DSR}}$ and $\overline{\text{CTS}}$. The modem control input $\overline{\text{DCD}}$ is accessible through channel A. Use of each individual modem control line is dependent on compatible facilities within the external equipment.

Generation of the output modem control lines and status monitoring of the input modem control lines is directly under program control. The SIO can be programmed to generate an interrupt to the CPU every time a transition occurs on any of the input control lines.

RS232C Connector Detail

The definition of the available RS232C connections provided by the 25-way D-type connector are detailed in the following table. The RS232C voltage levels are defined below. The line idle condition is indicated by continuous marking.

1. Line marking; Between —3V and —15V relative to 0V.
2. Line spacing; Between +3V and +15V relative to 0V.

RS232C Connector Pin Definition

Pin	Description	Pin	Description
1	Frame Ground	9 to 14	N.C.
2	Tx Data	15	Tc (input)
3	Rx Data	16, 18 ,19	N.C
4	RTS	17	Rc (input)
5	CTS	20	DTR
6	DSR	21 to 23	N.C.
7	0V (Signal GND)	24	TxClk (output)
8	DCD	25	N.C.

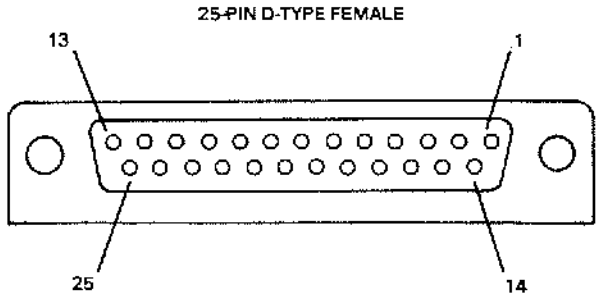


Figure 2. RS232C Connector Detail.

Channel B Programming Details

The following paragraphs detail the format of the Write Registers for setting channel B to operate in the asynchronous mode and also the various options available for controlling the transfer of data over the serial link. For details of programming the channel for the various synchronous communication modes, reference should be made to the appropriate Zilog documentation.

One other parameter is programmed through channel B. This is the base address for the SIO interrupt vector.

The address locations utilised by channel B and the method for writing data to the Write Registers are described in a previous section under the heading "Processor Interface".

Copy Registers

As with channel A, the BIOS also keeps a copy of certain SIO registers within channel B which may be of use to programmers. These are the registers WR1, WR3, WR4 and WR5.

The reason for keeping copies is to prevent contention arising between the BIOS and an application programmer when updating write-only registers. The mechanism for writing to the SIO registers (as described in the Keyboard copy registers) should also be adopted for channel B.

The offset from the double-word pointer for the channel B copy registers are as follows:

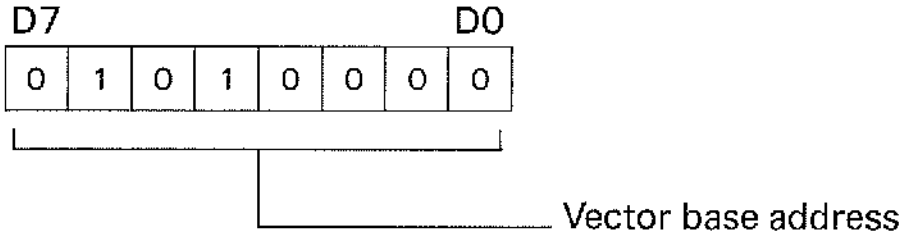
Write Register 1	00H
Write Register 3	01H
Write Register 4	02H
Write Register 5	03H

Setting the Base Vector

This is normally set up by the BIOS during an initialisation routine. The routine initialises the base vector to 50H by writing this value to channel B Write register 2.

The status affect vector bit in Write Register 1 of channel B (D2) also has to be set to enable an interrupt condition to modify the base vector.

Write Register 2



Asynchronous Communications

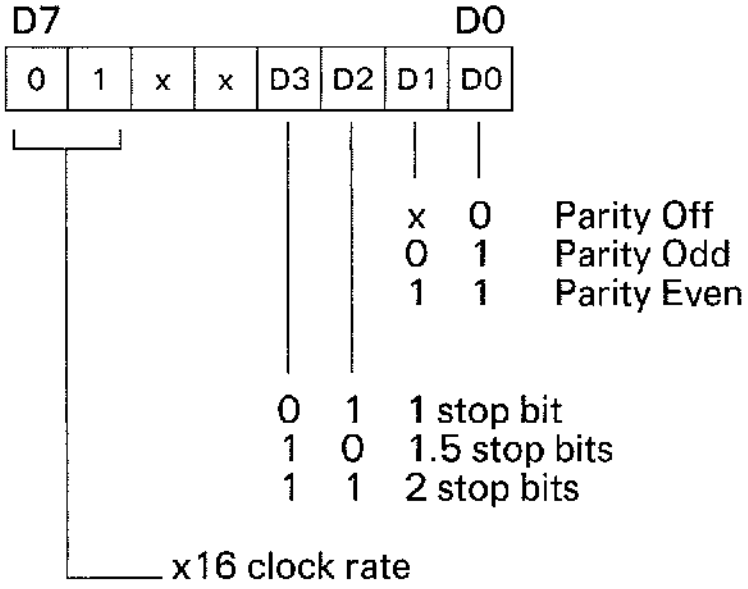
To receive and transmit data in the asynchronous mode, the following conditions have to be determined:

1. The transmit and receive baud rates.
2. The character length.
3. The number of stop bits in the transmit character (1 stop bit is always expected in the receive data).
4. The sense of the parity, if any.
5. The interrupt mode.
6. The line protocol.

Note: The modem control line DCD from the RS232C interface is connected to DCDA of channel A.

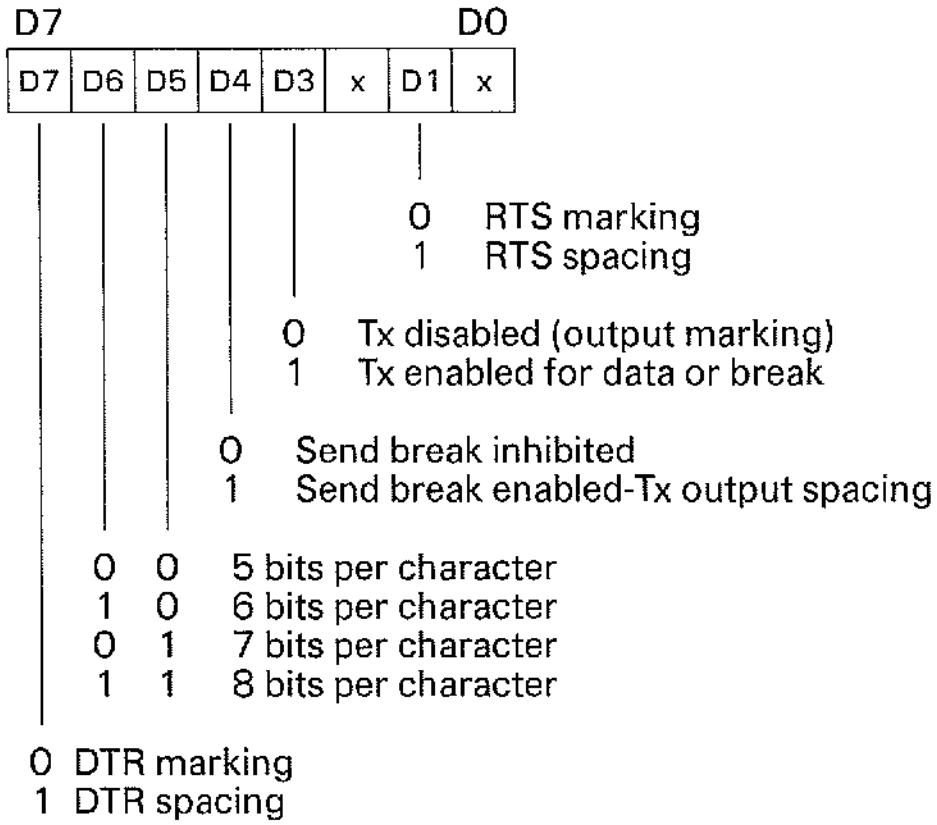
Write Register 4

This register determines; the number of stop bits in the transmit data, the sense of the parity (if any) in the transmit data and the expected parity in the receive data, and the multiplier applied to both the transmit and receive input clock rates prior to setting the baud rates.



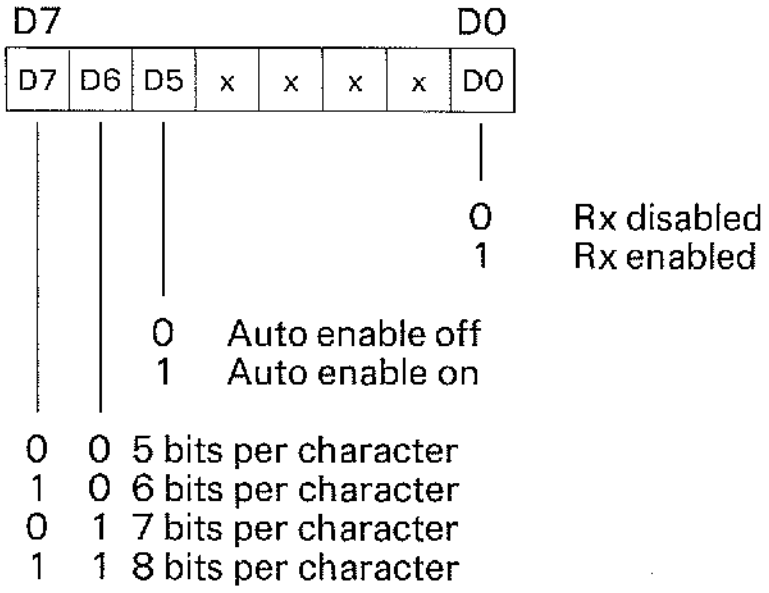
Write Register 5

This register determines the transmit character length, controls the modem outputs RTS and DTR, allows the programmer to enable/disable transmission, and also generate a break in transmission.


D7
D0
D7 D6 D5 D4 D3 x D1 x
0 RTS marking
1 RTS spacing
0 Tx disabled (output marking)
1 Tx enabled for data or break
0 Send break inhibited
1 Send break enabled-Tx output spacing
0 0 5 bits per character
1 0 6 bits per character
0 1 7 bits per character
1 1 8 bits per character
0 DTR marking
1 DTR spacing

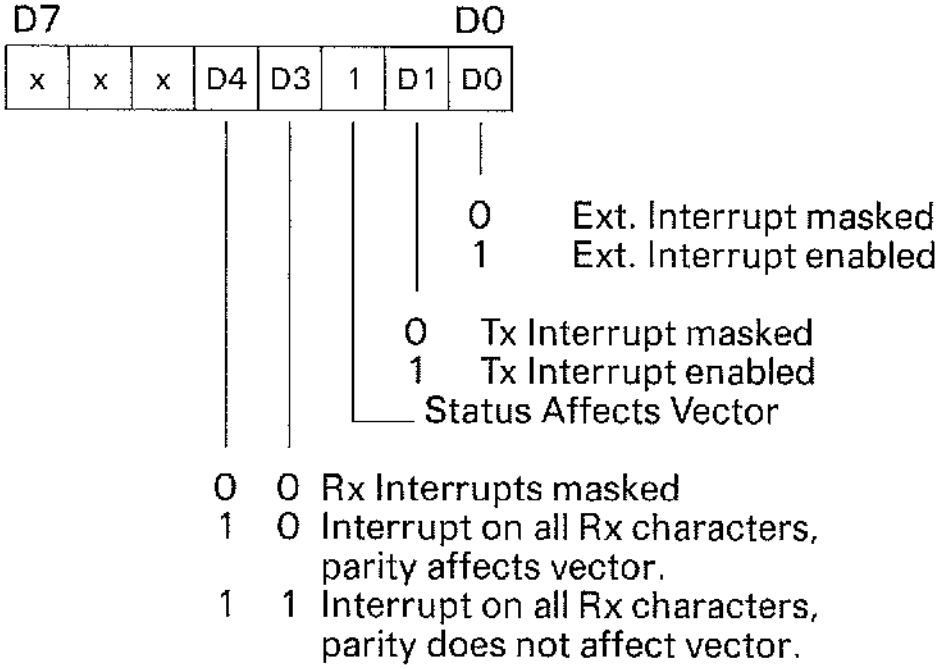
Write Register 3

This register determines the number of receive bits assembled to form a character, allows the programmer to enable/disable reception and also selects whether the modem control lines CTS and DSR control data transmission and reception over the serial link (Auto Enable).


D7
D0
D7 D6 D5 x x x x D0
0 Rx disabled
1 Rx enabled
0 Auto enable off
1 Auto enable on
0 0 5 bits per character
1 0 6 bits per character
0 1 7 bits per character
1 1 8 bits per character

Write Register 1

This register enables/disables the transmit, receive and external/status interrupts.


D7
D0
x x x D4 D3 1 D1 D0
0 Ext. Interrupt masked
1 Ext. Interrupt enabled
0 Tx Interrupt masked
1 Tx Interrupt enabled
Status Affects Vector
0 0 Rx Interrupts masked
1 0 Interrupt on all Rx characters, parity affects vector.
1 1 Interrupt on all Rx characters, parity does not affect vector.

SIO Pin Detail

The two SIO communication channels are designated channel A (used for the keyboard/mouse link and sound) and channel B (used for the RS232C link). On the block diagram at the beginning of the chapter, connections to channel A are denoted by the suffix A (e.g. TxDA), and to channel B by the suffix B (e.g. RxDB). All the remaining inputs and outputs of the SIO are connections to the processors. A description of each pin is detailed on the following pages.

System Connections

D0 to D7	Data bus, used to transfer data and commands between the processors and the SIO.
$\overline{\text{IORQ}}$	Input/Output Request. Control input, active state logic low, derived by combining the read and write commands from the system control bus ($\overline{\text{AIOWR}}$ and $\overline{\text{IORC}}$) using a series of logic gates. Used in conjunction with $\overline{\text{RD}}$ and the address inputs B/$\overline{\text{A}}$, C/$\overline{\text{D}}$ and $\overline{\text{CE}}$ to control the transfer of data and commands between the SIO and the CPU. An active state on $\overline{\text{IORQ}}$ indicates a transfer operation on the data bus; $\overline{\text{RD}}$ signifies the direction of data transfer (see below). $\overline{\text{IORQ}}$ also has another function. When set active at the same time as M1 is set active, it acts as an acknowledgement of a SIO interrupt, causing the SIO to release an interrupt vector onto the data bus.
$\overline{\text{M1}}$	Machine Cycle 1. Control input, active state logic low. Function as described above.

$\overline{RD}$	Read. Control input. Used in conjunction with $\overline{IORQ}$ and the address inputs B/$\overline{A}$, C/$\overline{D}$ and $\overline{CE}$ to control the transfer of data and commands between the SIO and processing elements. If both $\overline{IORQ}$ and $\overline{RD}$ are set low and a valid address is set up, the direction of data/command transfer is from the SIO (read operation). If $\overline{IORQ}$ is set low and $\overline{RD}$ is set high, with a valid address set up, the direction of data/command transfer is to the SIO (write operation).
$\overline{CE}$	Chip Enable. Address input, active state logic low. When active, indicates that the SIO is selected for a data/command transfer operation.
B/$\overline{A}$	Channel B/Channel A select. Address input. A logic low on B/$\overline{A}$ with $\overline{CE}$ also set low indicates that the data/command transfer operation involves channel A. A logic high on B/$\overline{A}$ with $\overline{CE}$ set low indicates that the data/command transfer operation involves channel B.
C/$\overline{D}$	Control/Data select. Address input. A logic low on C/$\overline{D}$ with $\overline{CE}$ also set low indicates that the information on the data bus is interpreted as data. A logic high on C/$\overline{D}$ with $\overline{CE}$ set low indicates that the information on the data bus is interpreted as commands.
CLK	System clock input. 2.33 MHz clock with a 50% duty cycle for internal timing within the SIO.
$\overline{RESET}$	System reset. Input, active low. When active, disables the receive and transmit sections of both channels and sets the transmit lines to the marking condition, requiring the registers within the SIO to be re-initialized before performing data transfer operations.
$\overline{INT}$	Interrupt request. Output, active low. Set active when an interrupt condition is detected internally within the SIO.

Channel A Connections

RxDA	Receive Data. Serial data input for data from the Keyboard and mouse. Decoded infra-red signals supplied via the IR Receiver Board.
TxDA	Transmit Data. Serial data output to the sound generator amplifier.
RxCA	Receiver Clock for channel A. Sets the receive rate of the internal receiver circuitry. Derived from the incoming IR data stream supplied from the IR Receiver Board.
TxCA	Transmitter Clock for channel A. Sets the transmit rate and thus controls the sound output frequency for the sound generator circuit.
$\overline{\text{SYNCA}}$	Synchronisation. Control output used to generate a hardware system reset. Everytime the SIO receives a sync character, the SIO pulses this pin low. When the Keyboard reset key is pressed, the keyboard transmits a contiguous stream of sync characters which results in pulses being produced in a regular 50% duty cycle. This causes a capacitor to gradually discharge. If the key is held down for approximately one second, the capacitor is discharged sufficiently to change the state of a trigger circuit which causes a system reset. This mechanism is discussed more fully in the System Detail chapter.
$\overline{\text{DCDA}}$	Data Carrier Detect A. Control input from the RS232C interface via a line receiver, connected to the Data Carrier Detect line — DCD. When used with a modem, the active state on DCD indicates that the modem has detected data sent to it. The input sets a control bit within an internal register according to the state of $\overline{\text{DCDA}}$ and also has the facility to generate an interrupt request to the CPU on every DCD transition.
$\overline{\text{RTSA}}$	Request to send. Used, in conjunction with $\overline{\text{DTRA}}$, as a general purpose output to select the volume level for the audio amplifier (as described in the Sound Generation chapter). The logic state on $\overline{\text{RTSA}}$ is controlled by software, following the state of an associated control bit within an internal register.

$\overline{\text{DTRA}}$	Data terminal ready. Used, in conjunction with $\overline{\text{RTSA}}$, as a general purpose output to select the volume level for the audio amplifier (as described in the Sound Generation chapter). The logic state on $\overline{\text{DTRA}}$ is controlled by software, following the state of an associated control bit within an internal register.
$\overline{\text{CTSA}}$	Clear to send. Control input, connected to the Busy control line from the Parallel Printer Port. This is normally programmed as a general purpose input, to generate an interrupt whenever a logic transition occurs on the printer Busy line.

Channel B Connections

TxDB	Transmit Data. Serial data output to the RS232C interface via a line driver. A mark is indicated by logic high on TxDB and a space, by logic low.
RxDB	Receive Data. Serial data input from the RS232C interface via a line receiver. A mark is indicated by a logic high and a space, by logic low.
TxCB	Transmitter Clock. Input used to determine the transmit baud rate of the RS232C channel. When operating in asynchronous mode, a facility exists within the SIO to divide the input clock frequency internally, prior to being used to set the baud rate. The divider is controlled by software and allows serial data to be transmitted at a rate of 1, 1/16th, 1/32nd or 1/64th of the rate supplied to TxCB. In synchronous modes, this facility is not available, so that TxCB corresponds directly to the transmit baud rate.
RxCB	Receiver Clock. Input used to set the receive baud rate of the internal data receiver circuitry. In asynchronous modes, the receiver clock is divided by the same divisor programmed for the transmitter clock. In synchronous modes, RxCB directly sets the receive baud rate of the data receiver circuitry.
$\overline{\text{DCDB}}$	Data Carrier Detect B. Control input from the RS232C interface via a line receiver, connected to the Data Set Ready line — $\overline{\text{DSR}}$. When used with a modem, the active state on $\overline{\text{DSR}}$ indicates that the modem has data to send. The input can be programmed to operate in one of two modes. The first mode allows $\overline{\text{DSR}}$ to act as the enable control line to the receive section of channel B. The second mode sets a control bit within an internal register according to the state of $\overline{\text{DSR}}$ and also has the facility to generate an interrupt request to the CPU, every $\overline{\text{DSR}}$ transition.

$\overline{\text{DTRB}}$	Data Terminal Ready. DTR control output to the RS232C interface via a line driver. When used with a modem, the active state (logic low) indicates that the SIO is ready to start handling data. $\overline{\text{DTRB}}$ is controlled by software, following the logic state of an associated control bit within an internal register.
$\overline{\text{RTSB}}$	Request to send. RTS control output to the RS232C interface via a line driver. When used with a modem, the active state (logic low) indicates that the SIO has data ready to send. $\overline{\text{RTSB}}$ is controlled by software, following the logic state of an associated control bit within an internal register.
$\overline{\text{CTSB}}$	Clear to send. CTS control input from the RS232C interface via a line receiver. When used with a modem, the active state (logic low) indicates that the modem is ready to receive data. $\overline{\text{CTSB}}$ can be programmed to operate in one of two modes. The first mode allows $\overline{\text{CTSB}}$ to act as the enable control line to the transmit section of channel B. The second mode sets an associated control bit within an internal register according to the state of $\overline{\text{CTSB}}$ and also has the facility to generate an interrupt request to the CPU, every time a transition occurs on $\overline{\text{CTSB}}$.

Printer Interface 2.8

Contents

Introduction

Details
- General
- Data Transfers
- Connector Detail
- Address Allocation

Programming Considerations
- Data Port
- Printer Status
- Data Strobe

Illustrations

1. Printer Interface
2. Centronics Connector Detail

Introduction

The Parallel Printer Port is designed to drive printers and plotters with a Centronics parallel interface.

The Printer Port has a 36-way, male connector, located on the System Board flush with the rear panel of the Systems Unit. The block diagram in Figure 1 shows the main circuit components.

The connector is wired for eight data output lines, and two of the handshake signals that are supplied on the majority of Centronics compatible printers; Data Strobe, and Busy.

Details

General

The Printer Interface consists of:

1. An 8-bit latch, for the transfer of data bytes to the printer.
2. A control line for strobing data bytes into the printer (Data Strobe).
3. A printer status line (Busy/Not Busy) from the printer. This is wired to the Serial Input/Output (Z80 SIO) controller on the System Board.

The Busy signal (active state - logic high) as supplied from the printer is used as a multi-purpose status signal. Its' fundamental role is to indicate that the printer is unable to receive data. The F1 views the cause of no consequence, as no operator feedback is provided via the computer. It may be because the printer buffer is full, the printer has run out of paper, or any other error/fault status condition.

The Busy signal is wired to one of the input control lines of the Z80 SIO (CTSA). This would normally be used as a Modem control input. Any transition on the Busy line, whether from high to low (Not Busy), or low to high (Busy), causes the SIO to generate an interrupt to the CPU and produce an interrupt vector. The associated service routine then can check the current state of the Busy line by reading a status register in the SIO (Read Register 0).

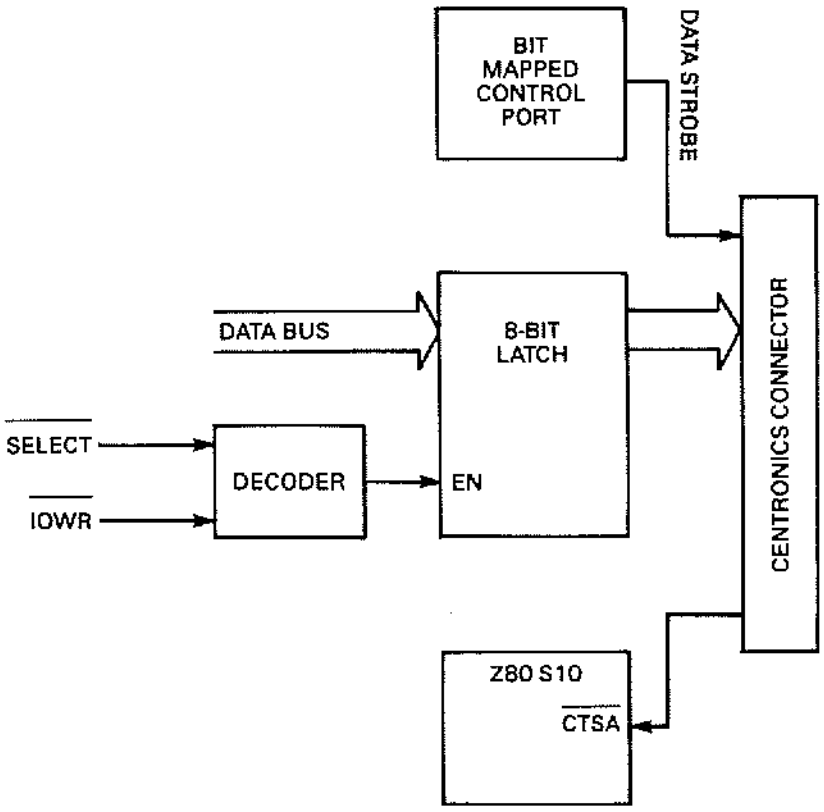


Figure 1. Parallel printer interface.

Data Transfers

The normal sequence of operations for the transfer of data is as follows.

The BIOS writes a byte of data to the data latch, and then reads the status register (Read Register 0) in the SIO Channel A to check on the status of the printer.

If the printer status is Busy, then the print routine waits for a Busy to Not Busy transition. The transition causes the SIO to generate an interrupt vector to the CPU.

When the printer status is set to Not Busy, the print routine sends a Data Strobe signal (low-to-high transition) to the printer interface. This latches the data byte stored at the outputs of the latch into the printer.

The process is then repeated for each data byte.

Connector detail

Pin	Description	Pin	Description
1	Data strobe	19	0V
2	D0	20	0V
3	D1	21	0V
4	D2	22	0V
5	D3	23	0V
6	D4	24	0V
7	D5	25	0V
8	D6	26	0V
9	D7	27	0V
10	N.C.	28	0V
11	Busy	29	0V
12	N.C.	30	0V
13	N.C.	31	N.C.
14	Link selection *	32	N.C.
15	N.C.	33	N.C.
16	0V	34	+ 12V out via 10 ohm resistor
17	Ground	35	Link selection **
18	Link Selection **	36	N.C.

* Normally open circuit, 0V with link fitted

** Normally open circuit, + 5V with link fitted

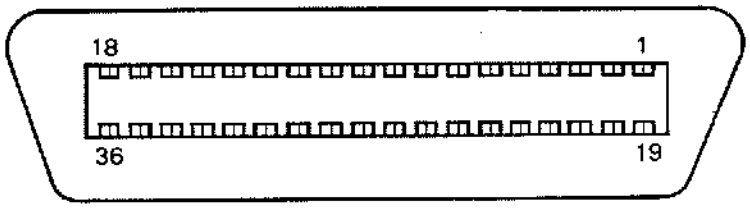


Figure 2. Centronics Connector pin detail

The definition of the control outputs to and from the printer connector is detailed below.

D0 to D7	8-bit data output
Data Strobe	Output timing signal. Used to latch the data into the printer. Normally at logic high. Positive edge of logic low pulse indicates printer data is valid.
Busy	Input signal. Logic high state indicates that the printer is unable to receive any data.

Address allocation

The system software views the printer interface as an array of ports in the System I/O Space.

The port addresses are defined by the chip select and other System Address Bus connections to the data output latch, the control port latch and the SIO.

Address	Port	Data	Access
00H	Printer	8-bit print code	Write only
0FH	Data Strobe	FFH to set low 00H to set high	Write only
22H	Busy	SIO Read Register 0 (Bit D5)	Read only
22H	Interrupt enable	SIO Write Register 1 (Bit D0)	Write only

Programming Considerations

Data Port

The printer data port is located at address 00H in the system I/O space. Writing data to this address latches the data through to the Centronic port outputs.

Printer Status

The Busy status line of the printer is wired to the CTSA input of the Z80 SIO. The programmer is able to directly monitor the state on this line by reading bit D5 in Read Register 0 of the SIO (Channel A). This bit indicates the inverse of the state on the CTSA input, i.e.

Bit D5 high = Not Busy

Bit D5 low = Busy

The SIO can also be programmed to generate an interrupt and supply an interrupt vector to the CPU on any signal transition on the control inout (Busy to Not Busy and Not Busy to Busy). This is normally set up by the BIOS.

The associated interrupt service routine is identified by the vector 5AH. Enabling the interrupt is achieved by setting bit D0 in Channel A Write Register 1 of the SIO high. The interrupt vector 5AH does not uniquely identify the cause of the interrupt, which may be caused by other external line/staus events than transitions on the Busy line (e.g. transitions on the DCD input from the RS232C interface).

Full details for programming the Z80 SIO are provided in the chapter headed "Serial Interface".

Data Strobe

The Data Strobe line is controlled by writing to a bit-wide port mapped in the system I/O space at odd address location 0FH. The port is wired to the LSB data line D8 on the high order section of the data bus. Writing FFH to the port sets the Data Strobe line low. Writing 00H to the port sets the Data Strobe line high.

Contents

Introduction

Details
- General
- Channel modes
- Clock rates
- Interrupts
- Channel usage
- Address allocation

Programming Considerations
- Initialisation
- Setting the base interrupt vector
- Channel 0: Expansion interrupts
- Channel 1: RS232C baud rate
- Channel 2: Sound frequency
- Channel 3: System clock
- Return from Interrupt Sequence

Illustrations

1. Counter/Timer

Introduction

The System Clock interrupt is generated on a regular cycle of 20 ms by the Z80 Counter/Timer Circuit (CTC). This integrated circuit is located on the System Board.

The Z80 CTC is a multi-purpose timing device, with four programmable counter/timer channels and a prioritised interrupt structure. The channels are numbered from Channel 0 to Channel 3, and are used to generate the following functions:

Channel	Usage
0	Expansion Bus interrupts.
1	Transmit and receive baud rate clocks for the RS232C communications interface.
2	The fundamental frequency for generating sound.
3	The System Clock interrupt.

Channel 0 is programmed to respond to interrupt requests from the Expansion Bus. Channels 1 and 2 can be programmed to produce timing signals which are used by the Serial Input/Output (SIO) controller for RS232C communications and sound generation, respectively. Channel 3 is used by the BIOS for the system clock.

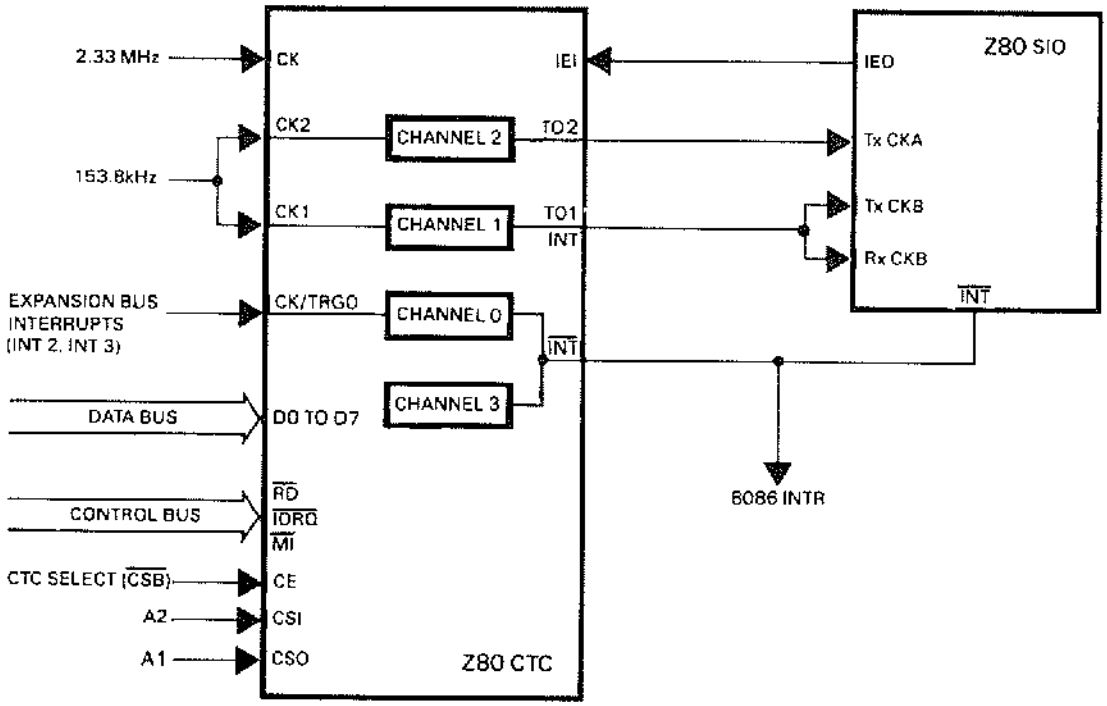


Figure 1. Counter/timer

Details

General

The CTC is organised internally as four, independent counter/timer channels, each with:

1. A clock/trigger input.
2. A zero count/timeout output.
3. A Control Register (Write only).
4. A Time Constant Register (Write only).
5. A Downcounter Register (Read only).
6. An interrupt generator.

There is also a clock input for the CTC as a whole which is used for internal timing within the CTC and can also be used as a clock source for decrementing the downcounter registers.

Channel modes

Each channel operates independently in one of two modes: counter mode, or timer mode.

In counter mode, the value in the Downcounter Register is decremented each time a pulse is detected on the clock/trigger input.

In timer mode, the value in the Downcounter Register is decremented regularly at each CTC clock input (2.33 Mhz).

In either mode, when the value in the Downcounter Register has been decremented to zero, the CTC generates a pulse on the zero count/timeout output for that channel. The Downcounter is then reloaded from the Time Constant Register, and the count down is repeated.

Each channel can also be programmed to generate an interrupt to the CPU via the CTC's interrupt output, on reaching zero count state. The CTC can also produce an interrupt vector to specify the cause of the interrupt.

Each channel is programmed independently, by writing a command byte to the Channel Control Register and then loading an integer value into the Time Constant Register.

In counter mode, as soon as the Time Constant Register has been loaded, it's contents are transferred to the Downcounter Register, and the downcounter is decremented everytime a clock pulse appears on the clock/trigger input for the channel.

In timer mode, the count down cycle is programmed to begin either as soon as the Time Constant is loaded, or when the first pulse is detected on the clock/trigger input.

The count down and reload sequence is repeated continuously until a reset command is written to the channel Control Register.

Clock rates

In counter mode, the clock pulse for decrementing the count is taken from the clock/trigger input for the channel.

A channel can be set to decrement after either the rising or the falling edge of the clock/trigger pulse, but the decrement itself is synchronised with the next rising edge of the CTC timer clock.

In timer mode, the clock pulse for decrementing the count is taken from the CTC's clock input, which runs at 2.33 MHz. This clock rate is prescaled within the channel, to divide the output clock frequency either by 16 or 256.

Interrupts

The CTC is connected in a daisy chain interrupt configuration, with the Z80 SIO on the System Board. The interrupt output lines of both devices are wired to the single interrupt input (INTR) of 8086 CPU. The Z80 SIO has higher priority interrupt structure than the CTC. This is set by the hardwired connection IEO (Z80 SIO) to IEI (Z80 CTC).

If an interrupt condition occurs within the CTC, it can only initiate the associated service routine providing there are no Z80 SIO interrupts queued and no Z80 SIO service routines in progress. These two states are indicated by the SIO setting it's IEO output low.

When an interrupt condition occurs within the CTC and the IEI input is set high, the CTC generates an interrupt request to the CPU. The CPU then performs an interrupt acknowledge cycle, which releases a modified pre-programmed interrupt vector onto the data bus to indicate the cause of the interrupt. This is then used by the CPU to vector to the appropriate service routine.

At the end of the interrupt cycle, the programmer has to send a two-byte RETI (RETurn from Interrupt) command to terminate the interrupt cycle and allow any lower priority interrupts to be serviced.

Within the CTC, the channels also have an interrupt priority. Channel 0 has the highest priority, followed by Channel 1 and then Channel 2, with Channel 3 having the lowest priority. Interrupt requests from a lower priority channel are queued until those at a higher priority have been cleared.

This method of prioritising interrupts is described in detail in the chapter headed Interrupt Control.

In either counter or timer mode, if interrupts are enabled on a particular channel, then an interrupt request is generated internally when the downcount reaches zero.

The pre-programmed interrupt vector is a base address for the interrupt vectors capable of being produced by the four CTC channels. When an interrupt is generated on one of the four channels, the base vector is modified by adding 0, 2, 4 or 6 to its value, according to whether the interrupt condition occured on Channel 0, 1, 2 or 3. This modified vector is accessed by the CPU during the interrupt acknowledge sequence.

The base address is 60H and is loaded by the BIOS, by writing this value to the Channel 0 Control Register.

Channel usage

Channel 0 is programmed by the BIOS to operate in counter mode, and is used to vector interrupt requests caused by interrupts occuring on the Expansion Bus (control lines INT2 and INT3).

These two control lines normally produce a level-sensitive interrupt and are connected to the inputs of a NAND gate to form a single interrupt line to the trigger input of channel 0 (CK/TRG0). This line is also gated with a clock signal, prior to being supplied to the CTC, to enable the CTC to be triggered by the "later" Expansion Bus interrupt when both control lines become active.

When an Expansion bus interrupt occurs, the clock signal causes a positive edge to be produced on the Channel 0 clock/trigger input. This causes the Downcounter to be decremented from 1 to 0 (the value of 1 is the pre-programmed Time Constant, set by the BIOS), and the Downcounter to be reloaded from the Time Constant register.

On reaching zero, an internal interrupt is generated which modifies the base interrupt vector to 60H and an active interrupt request is supplied to the CPU (providing no higher priority interrupts are currently in service). The modified interrupt vector is accessed by the CPU, during it's interrupt acknowledge cycle.

Further positive edges produced on this clock/trigger input are ignored until a RETI sequence is sent to the CTC, to terminate the Expansion Bus interrupt sequence.

Further details on how to use Expansion Bus interrupts are provided in the Expansion chapter.

Channels 1 and 2 can be programmed to operate in either counter or timer mode, to provide clock inputs to the Z80 SIO for the RS232 interface and for generating sound. In some cases the timer mode clock (2.33 MHz), used with a period prescaler of x 16 or x 256, can generate a more accurate rate; in other cases, it is more convenient to use the counter mode clock/trigger inputs (153.8 KHz).

Channel 1 can be used to set the baud rate for both the Transmit and Receive clocks on the RS232C channel of the SIO.

Channel 2 generates a clock which sets the fundamental frequency of the pulses sent by the SIO to the audio output for sound generation.

Channel 3 operates in timer mode, and is programmed by the BIOS to generate a System Clock interrupt every 20 ms. Its clock/trigger input is wired to a display timing signal but is not currently used.

Address allocation

The system software views the Channel Control Register, the Time Constant Register, and the Downcounter of each channel as an array of ports in the System Input/Output Space.

The port addresses are defined by the chip select and other System Address Bus connections to the CTC, as follows:

Address	Channel	Register	Access
10H	0	Control/Time Constant	Write only
10H	0	Downcounter	Read only
12H	1	Control/Time Constant	Write only
12H	1	Downcounter	Read only
14H	2	Control/Time Constant	Write only
14H	2	Downcounter	Read only
16H	3	Control/Time Constant	Write only
16H	3	Downcounter	Read only

Programming Considerations

Initialisation

Each channel is initialised separately, by writing a command byte to its Control Register, followed by an integer value byte for its Time Constant Register. Both bytes have to be written in sequence to the same I/O address. The command byte is sent first, followed by the time constant byte, if required (bit 2 within the command byte is used to signify to the CTC whether the following byte is the time constant byte or not).

The command byte sets the channel mode and related options, enables or disables interrupts, and sets the count to occur on the rising or the falling edge of the input.

The Time Constant Register holds the value to be loaded into the Downcounter at the start of each count cycle. It can be overwritten during a count without affecting the current contents of the Downcounter. The new value is then loaded into the Downcounter at the next zero count.

The current Downcounter value can be read at any time without disturbing the count.

In counter mode, counting is initiated by the first clock pulse after the Time Constant Register has been loaded.

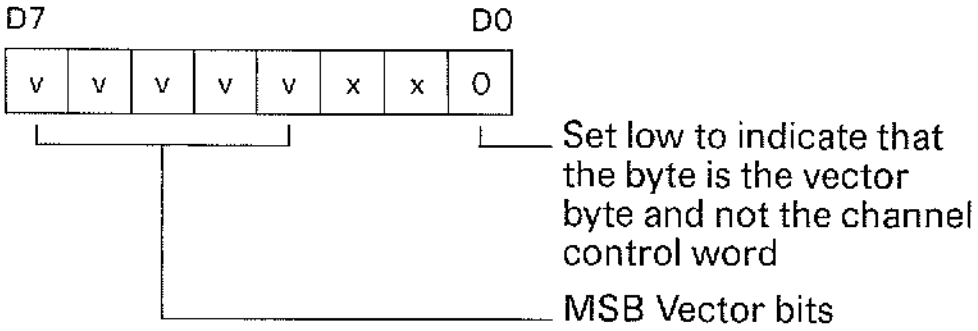
In timer mode, counting can be set to begin either when the Time Constant Register is loaded, or to be triggered by the next clock/trigger input. No action is taken on a clock/trigger input until the next rising edge of the CTC's 2.33 MHz clock input.

Counting continues uninterrupted until a reset command is written to the Control Register for the channel. After a reset, the Control and Time Constant Registers must be re-initialised.

Prior to using interrupts, the base address of the interrupt vector accessed by the CPU during an interrupt acknowledge sequence must be written to the CTC. This is achieved by writing a special command byte to Channel 0 of the CTC. This is the first byte normally written to the CTC during initialisation.

Setting the base interrupt vector

The BIOS sets the base address for interrupt vectors from the CTC, by writing the value (60H) to the Control Register port of Channel 0. The general format of the command byte to set the base address is as follows:



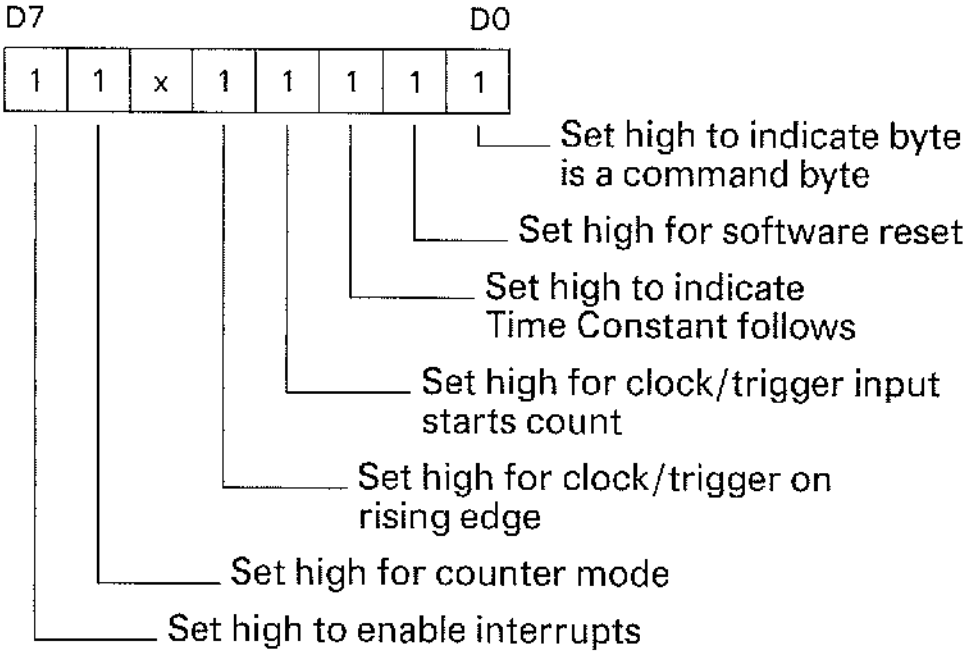
x = immaterial, program to zero.

Channel 0: Expansion Bus interrupts

Channel 0 is initialised by the BIOS to generate an interrupt vector on receiving an active interrupt request from the Expansion Bus.

This is achieved by writing two bytes in sequence to I/O address 10H. The first byte is the command to configure the channel. The second byte is the Time Constant value.

The format of the command byte is as follows:


D7
D0
1 1 X 1 1 1 1 1
Set high to indicate byte is a command byte
Set high for software reset
Set high to indicate Time Constant follows
Set high for clock/trigger input starts count
Set high for clock/trigger on rising edge
Set high for counter mode
Set high to enable interrupts

This command byte is written to the Channel 0 Control Register, to set the channel into counter mode with interrupts enabled.

The Time Constant value written to I/O address 10H by the BIOS is 01H.

The channel is thus set to count down from 1 to 0 and then generate an interrupt on receiving a positive edge on it's clock/trigger input.

Channel 1: RS232C baud rate

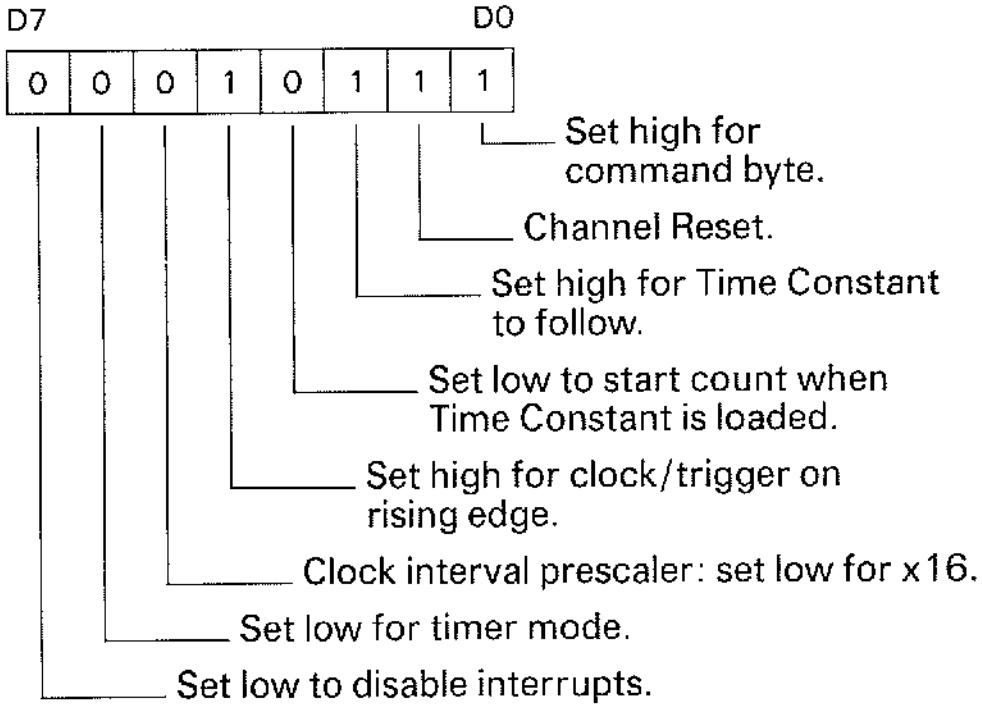
Channel 1 is initialised to generate the Transmit and Receive clock inputs for the RS232C interface formed by the Z80 SIO. These clocks have to be switched off if the SIO is to take its clocks from the external device connected to the RS232C interface. This is achieved by writing a reset command to Channel 1 with the "Time Constant follows" bit set (bit 2) and ***not*** sending the Time Constant byte (see below).

Channel 1 produces a pulse train for the Transmit and Receive clocks for the RS232C serial interface. It can be programmed to operate in either counter or timer mode, to provide clock inputs to the SIO at the required rate.

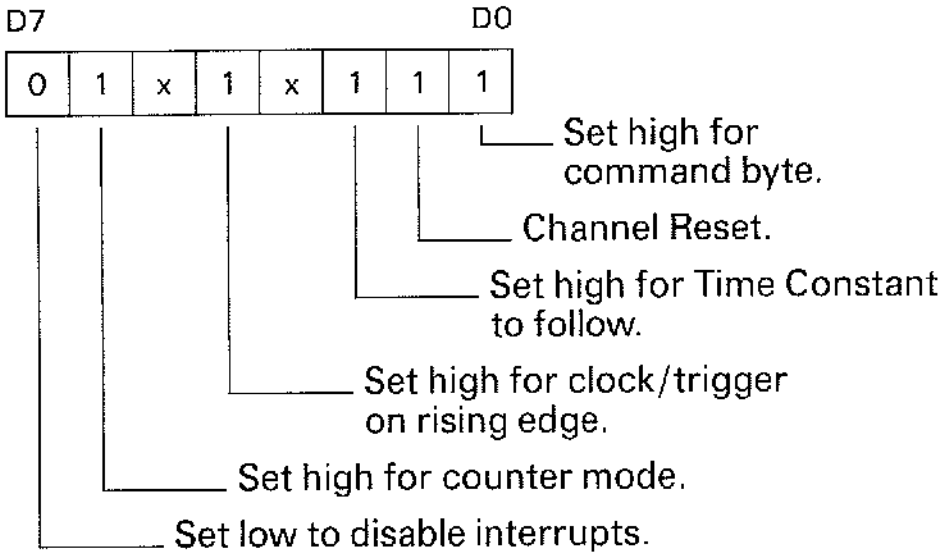
In some cases, the timer mode clock (2.33 MHz), used with a period prescaler of x 16, can generate a clock rate that is closer to the desired frequency; in other cases, it is more convenient to use the counter mode clock/trigger inputs (153.8 KHz).

In all cases, interrupts are disabled on this channel.

The format of the command byte for the timer mode is as follows:


D7
D0
0 0 0 1 0 1 1 1
Set high for command byte.
Channel Reset.
Set high for Time Constant to follow.
Set low to start count when Time Constant is loaded.
Set high for clock/trigger on rising edge.
Clock interval prescaler: set low for x16.
Set low for timer mode.
Set low to disable interrupts.

The format of the command byte for the counter mode is as follows:


D7
D0
0 1 x 1 x 1 1 1
Set high for command byte.
Channel Reset.
Set high for Time Constant to follow.
Set high for clock/trigger on rising edge.
Set high for counter mode.
Set low to disable interrupts.

The command byte is written to the Channel 1 Control Register. This then followed by an integer value, which is written to the Time Constant Register to set the output clock rate.

Listed below are the count values for programming Channel 1 to generate some of the commonly used baud rates. These are the values used for asynchronous communications when the Z80 SIO is programmed to internally divide the incoming baud rate clock inputs by 16.

Equivalent Baud Rate	Count value (Hex)	Channel mode
50	C0	Counter
75	80	Counter
110	53	Timer
134.5	44	Timer
150	40	Counter
300	20	Counter
600	10	Counter
1200	08	Counter
1800	05	Timer
2400	04	Counter
4800	02	Counter
9600	01	Counter

The formulae for calculating the required clock rate in each of the two CTC channel modes are given below.

In counter mode, using the 153.8 KHz clock, the baud rate (in bauds) is given by the following formula:

153800/(16 x Time Constant value) =
9613/Time Constant

Note: The divider factor of 16 is provided by the Z80 SIO when operating in asychronous mode (not the CTC).

In timer mode, using the 2.333 MHz clock with a x 16 clock period prescaler set in the CTC channel, the baud rate (in bauds) is:

2333333/(16 x 16 x Time Constant) =
9114.6/Time Constant

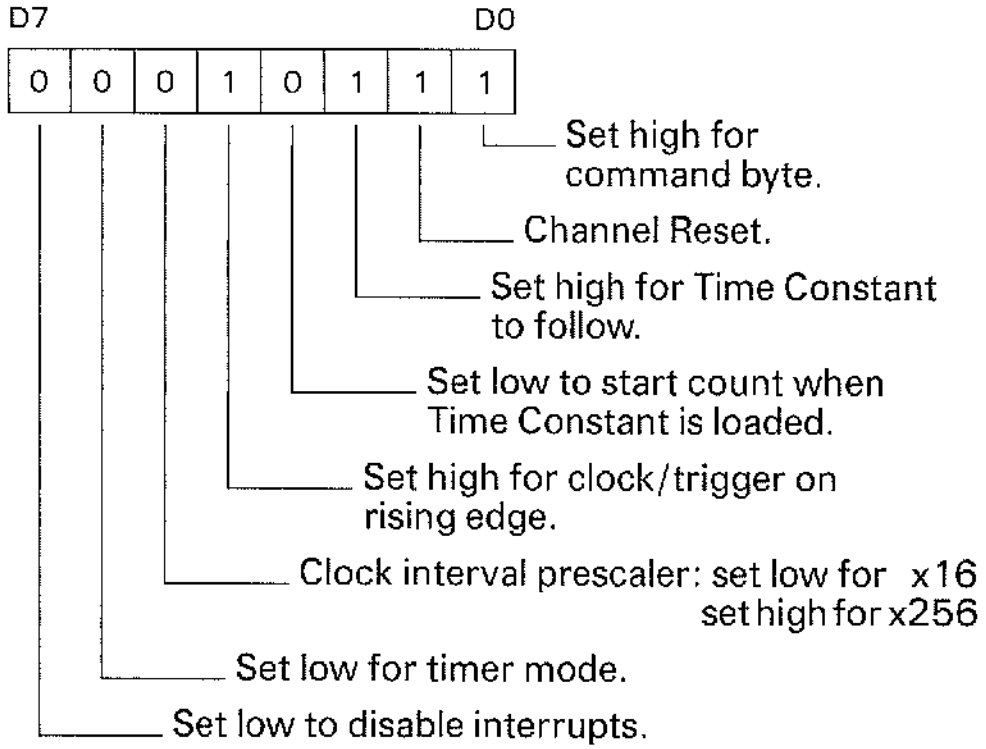
Note: One of the divider factors of 16 is provided by the pre-scaler; the second is provided by the Z80 SIO when operating in asychronous mode.

Channel 2: Sound frequency

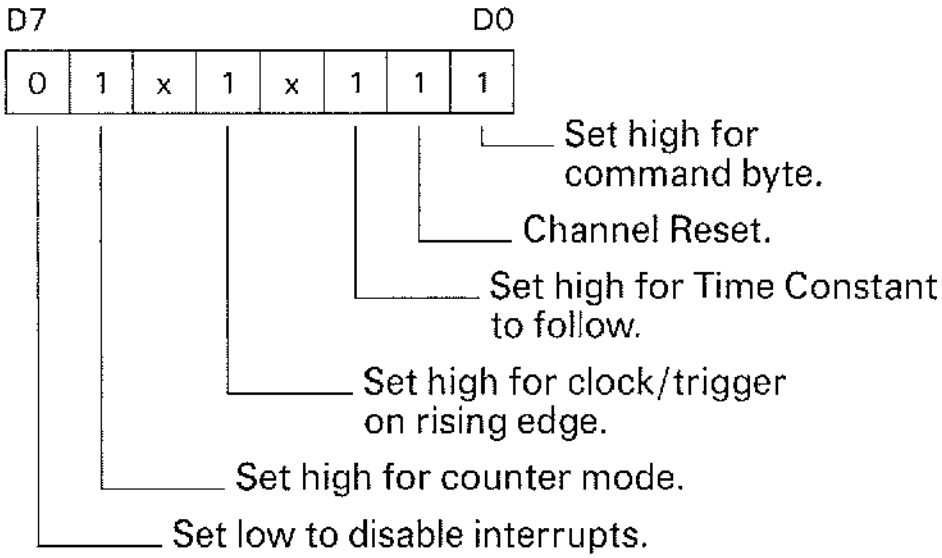
Channel 2 is used to generate the clock for the fundamental frequency at which bit pulses are transmitted by the SIO to the audio output circuitry for sound generation.

Channel 2 can be programmed to operate in either counter or timer mode, to provide clock inputs to the SIO at a particular rate. In practice, this fundamental frequency is limited by the frequency range of the loudspeaker which is from 600 Hz to 3 kHz.

The format of the command byte for the timer mode is as follows:


D7
D0
0 0 0 1 0 1 1 1
Set high for command byte.
Channel Reset.
Set high for Time Constant to follow.
Set low to start count when Time Constant is loaded.
Set high for clock/trigger on rising edge.
Clock interval prescaler: set low for x16 set high for x256
Set low for timer mode.
Set low to disable interrupts.

The format of the command byte for the counter mode is as follows:



The command byte is written to the Channel 2 Control Register. This then followed by an integer value, which is written to the Time Constant Register to set the output clock rate.

The formula for the clock frequency output (Fck in Hz) in timer mode is as follows.

Fck = 2333333/(Prescaler x Time Constant)

The formula for the clock frequency output (Fck in Hz) in counter mode is as follows.

Fck = 153800/Time Constant

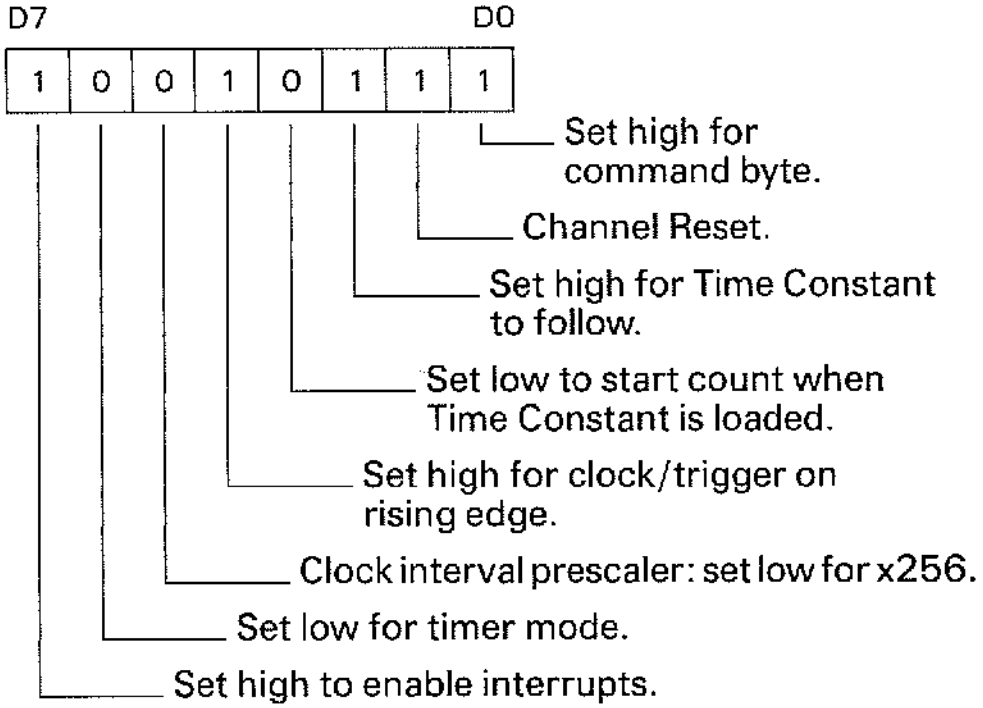
Further details are given in the chapter on Sound Generation.

Channel 3: System Clock Interrupt

Channel 3 is initialised by the BIOS to generate an interrupt every 20 ms to act as a System Clock.

The interrupt service routine reads the current value in the Downcounter Register, to determine when the interrupt was generated. It then updates the BIOS clock and performs other timer-related routines.

The format of the command byte is as follows:


D7
D0
1 0 0 1 0 1 1 1
Set high for command byte.
Channel Reset.
Set high for Time Constant to follow.
Set low to start count when Time Constant is loaded.
Set high for clock/trigger on rising edge.
Clock interval prescaler: set low for x256.
Set low for timer mode.
Set high to enable interrupts.

This command byte is written to the Channel 3 Control Register to set the channel into timer mode, with a clock prescaler of x 256. Then an integer value of 182 decimal (B6H) is written to the Time Constant Register, to set the output clock rate at 50.08 Hz.

Return from Interrupt Sequence

At the end of an interrupt service routine, a two-byte RETurn from Interrupt (RETI) command, has to be issued to the CTC to allow other interrupts to be serviced. The two bytes in the command are EDH, followed by 4DH.

The RETI sequence is generated by writing these two command bytes in sequence to port 30H in the System I/O Space.

Sound Generation 2.10

Contents

Introduction

Details
- General
- Generating Sound
- Address allocation

Programming Considerations
- General
- Initialisation
- Simple tones
- Complex sounds

Illustrations

1. Sound generation

Introduction

The sound generator is a single channel "device". It can be programmed to generate simple audio tones, audio noise, or much more complex waveforms in the form of synthesised sounds, over the frequency range 600 Hz to 3 kHz.

The programmer has independent control over the frequency, tone, volume, and duration of the audio output produced from the sound generator.

The "device" is also used by the BIOS to generate:

1. The keyboard click, whenever a key is depressed.
2. The bell tone, which is used as a warning signal in a number of applications.

Instead of using a proprietary sound generator chip as the source for driving the internal speaker, the sound generator "device" is formed by two programmable elements which are also employed for a variety of other purposes. These are the two Zilog devices: the Z80 Serial Input/Output (SIO) controller, and the Z80 Counter/Timer Circuit (CTC).

The other functions that these two devices are used for are discussed in detail in the chapters Serial Interface and Timer.

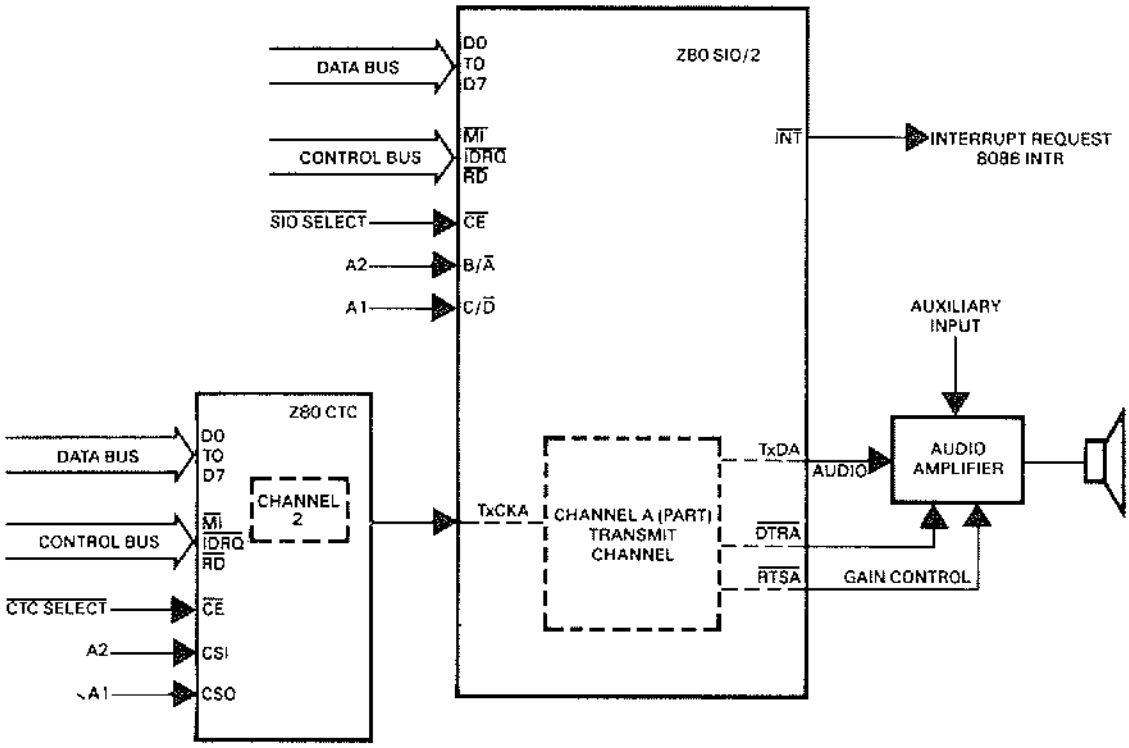


Figure 1. Sound generation.

Details

General

The sound generator circuitry consists of a loudspeaker (16 Ohm, 0.4 W) driven by an audio amplifier. It is fed with a pulsed signal from the data transmit output (TxDA) of Channel A in the Z80 SIO.

The channel is programmed to operate in the byte oriented synchronous mode, Monosync. This matches the requirements of the data receive input of Channel A, which is used as the input for data from the infra-red keyboard.

The block diagram in Figure 1 shows the main functional components.

Programmable facilities within the SIO allow the audio output to be switched off and on, and also provide the means of controlling the audio output level.

On/off switching, which sets the duration of an output signal, is controlled by writing to a register within the SIO. A bit setting within the register enables/disables the transmission from the SIO.

The volume is controlled by writing to a SIO register to set the state of two programmable outputs that are normally used as modem control lines; RTSA and DTRA. The combination of logic states on these two outputs allows the output volume to be set to one of four discrete levels.

The frequency of the pulsed audio output signal is set by the input clock frequency to Channel A of the SIO (via TxClkA). This clock frequency is generated by the Z80 CTC.

A bandpass filter in the amplifier section (25 Hz to 7 kHz) attenuates the high frequency components in the audio output signal, thereby smoothing the pulsed square wave signals from the SIO. The loudspeaker then further limits the frequency range of the signal from 600 Hz to 3 kHz.

The speaker is wired to a 2-pin Molex connector located half way down the left side of the System Board. An auxiliary audio input is provided via another 2-pin Molex connector (next to the internal Expansion Slot), to allow other devices to use the loudspeaker. It's nominal maximum input level is 50mV.

Generating Sound

Sound can be generated by one of two methods.

The first method can be used to produce audio tones and simple noises.

The second method can be used to produce much more complex waveforms and synthesised sounds.

The first method requires programming channel A of the SIO to repeatedly transmit a sync byte to the audio amplifier. This is the sync header byte of a normal monosync transmission. The regularity of the bit pattern within the 8-bit sync byte controls the purity of the output tone. The frequency of the output is controlled by timer pulses generated by Channel 2 of the CTC. These pulses act as the baud rate control for the synchronous transmission.

The second method requires programming channel A of the SIO to transmit a series of data bytes, as in a standard monosync transmission. Complex sounds can be produced by varying the data sent to the SIO in combination with varying the frequency of the output via control of the CTC.

The two methods require slightly different programming sequences. These are described in the section Programming Considerations.

The SIO Channel has an associated set of Write Registers to control its operations. Write Register 0 is used to reset the channel; for example, following an interrupt. Write Register 1 is used to enable particular types of interrupt.

Data for transmission (as required for the second mode described above) is written to the Channel A data port.

There is also a pair of Read Registers in Channel A. These record the current state of channel transmit and receive operations. Full details of all SIO registers are provided in the chapter on the Serial Interface.

The CTC has a Control Register and a Time Constant Register. The Control Register is programmed to set the required CTC mode. The Time Constant Register is loaded with a value relating to the period of the fundamental frequency of the sound output transmissions. Full details of these CTC registers are provided in the chapter on the Timer.

Address allocation

The system software views the control registers in the SIO and the CTC, and the data register in the SIO, as an array of ports in the System I/O Space.

The port addresses are defined by the chip select and other System Address Bus connections to the SIO and the CTC. The ports which are addressed in connection with sound generation are listed below. The usage of each register is discussed in the later section Programming Considerations.

Address	Channel	Register	Access
14H	CTC 2	Control/ Time Constant	Write only
20H	SIO A	Data (bytes for transmission)	Write only
22H	SIO A	Command (Write Registers)	Write only
22H	SIO A	Status (Read Registers)	Read only

Programming Considerations

General

The audio output can be programmed in two ways:

1. By the repetitive serial transmission of a single sync byte. This method can be used for generating simple tones.
2. By the interrupt-driven serial transmission of a series of data bytes. This method can be used for generating more complex waveforms.

The more regular the bit transitions in the transmitted bytes, the purer the tone. An irregular sequence of bit transitions, depending on its duration and frequency, can be used to generate anything from a click to a whine.

The audio parameters listed below can be programmed independently for either method of producing sound.

1. Waveform shape.
2. Frequency.
3. Volume.
4. Duration.

The waveform shape is determined by the bit pattern that is pulsed out to the audio amplifier. In mode one described above this is determined by the sync header byte written to the SIO. In mode two, it depends on the pattern of bytes written to the SIO for transmission.

The frequency of the bit pulses out of the SIO is set by programming Channel 2 of the CTC.

The volume can be set at one of four discrete levels by setting two control bits in SIO Write Register 5. These two bits control the logic levels on the Channel A outputs, DTRA and RTSA, to the pre-amp.

The duration of the output is controlled directly by enabling and disabling a control bit in SIO Write Register 5.

The SIO transmits a pulsed output to the amplifier in both modes. The characteristics of the amplifier and speaker, bandlimit the high frequency components of the pulsed waveform into the audio spectrum.

In mode one, by choosing a regular bit pattern for the sync byte (e.g. 4 1's for the most significant bits and 4 0's for the least significant bits) and programming an acceptable clock frequency into the CTC, the SIO produces a rough equivalent of a sinusoidal waveform.

Initialisation

The BIOS initialises SIO Channel A to operate in monosync mode to enable it to receive data from the keyboard. The data, write registers, port addresses, method of writing to the SIO, etc., for the initialisation procedure are detailed in the chapter on the Serial Interface.

The BIOS sets up channel A to receive/transmit data in monosync mode with an x1 baud rate clock. The programmer/BIOS then can control the sound output by writing to various registers in the SIO.

The following paragraphs detail the registers in the SIO and CTC for controlling the sound output in each of the two modes. They merely specify the registers and bits within the registers of interest. For details of how to access the registers, reference should be made to the appropriate chapter describing the device (Z80 SIO - Serial Interface chapter, Z80 CTC - Timer chapter).

Simple tones

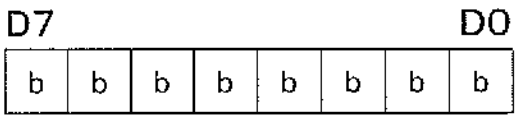
In this mode, the programmer has to write a sync byte to the SIO and set up the required fundamental frequency of the tone by writing to channel 2 of the CTC. He then can switch the "tone" on and control it's volume by writing to another register in the SIO.

The sync byte is repeatedly sent to the amplifier at the selected clock frequency until switched off.

Waveform Shape

The bit pattern of the sync byte is set by writing to Write Register 6 (WR6).

This is achieved by issuing two consecutive bytes to channel A of the SIO (port located at 22H in the system I/O space). The first byte is written to Write Register 0 to provide a pointer to select Write Register 6. The second byte contains the bit pattern as detailed below.



b = Bit settings as desired

Frequency

The frequency at which bits in the sync byte are transmitted to the audio amplifier is set by programming Channel 2 of the CTC, which is located at port address 14H in the System I/O Space.

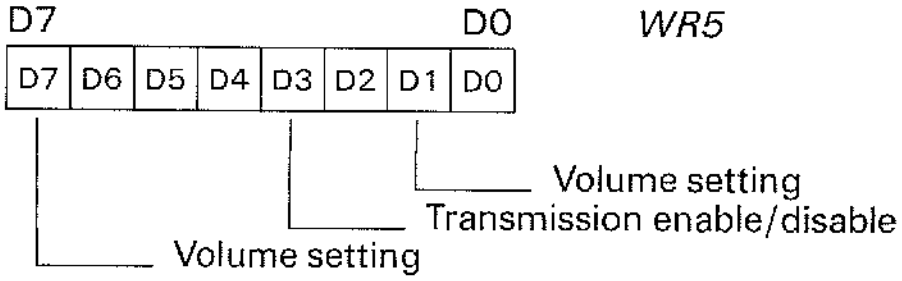
The channel is programmed by writing a command byte to reset the channel Control Register, followed by an integer value which is loaded into the Time Constant Register to set the frequency. This is fully described in the chapter on the Timer.

Volume

The volume is set at one of four discrete levels by programming a two-bit code on the SIO Channel A outputs, DTRA and RTSA. These two outputs are controlled by the settings of two associated control bits in Write Register 5.

This is achieved by issuing two consecutive bytes to channel A of the SIO (port located at 22H in the system I/O space). The first byte is written to Write Register 0 to provide a pointer to select Write Register 5. The second byte contains the bit settings. Care should be taken not to modify any of the other bits within the byte to avoid corrupting the operation of the SIO. (This is discussed in the Serial Interface chapter, under the heading of "Copy registers").

The bits within Write Register 5 which control the volume are as follows:



Loudness	D7	D1
Level 0	0	0
Level 1	0	1
Level 2	1	0
Level 3 (Full volume)	1	1

Duration

The duration of the output is controlled directly by clearing and setting control bit D3 in Write Register 5 of SIO Channel A.

This can be done at the same time as the volume is set, as described above.

Complex Sounds

Complex sounds can be produced by programming transmit channel A of the SIO to send a monosync transmission. The programmer has to define his desired waveform as a pulse train and then write each byte to the SIO under interrupt control.

Enabling the transmit interrupt

For complex sound synthesis, SIO Channel A must be set to generate an interrupt each time the transmit buffer is empty.

The interrupt for transmit buffer empty is enabled by setting control bit D1 in SIO channel A Write Register 1.

This is achieved by issuing two consecutive bytes to channel A of the SIO (port located at 22H in the system I/O space). The first byte is written to Write Register 0 to provide a pointer to select Write Register 1. The second byte is issued with D1 set. Care should be taken not to modify any of the other bits within the byte to avoid corrupting the operation of the SIO. (This is discussed in the Serial Interface chapter, under the heading of "Copy registers").

Waveform

Once the transmit interrupt is enabled, the SIO will generate an interrupt every time it's 8-bit transmit buffer is empty. The associated interrupt service routine has to be located at the address pointers specified by the interrupt type vector 58H.

The interrupt service routine has to write a data byte to the SIO Channel A data port, at port address 20H in the System I/O Space, every time an interrupt occurs.

If the interrupt request is not serviced in time for the next byte to be transmitted, then the transmissions default to repeatedly sending the sync byte to the audio amplifier, as in tone generation. Data transmissions resume as soon as there is another data byte to send.

Frequency, volume, and duration

These audio parameters are set in the same way for complex sound generation as they are for simple tone generation. Please refer to the previous subsection.

Disk Drive 2.11

Contents

Introduction

Details

General
Interface Details
Interface Connections (Outputs)
Interface Connections (Inputs)
Disk Drive Mechanism
Read/Write Heads
Head Positioning Mechanism
Head Load Mechanism
Sensors and Detectors
Drive Switch Settings
Drive Specification

Disks

General
Disk Precautions
Disk Insertion/Removal
Write Protecting
Disk Format

Illustrations

1. MicroFloppy Disk Drive
2. Interface block diagram
3. Connector location
4. Drive Select Control
5. Drive Motor Control
6. MicroFloppy Disk

Introduction

This chapter provides information on the disk drive fitted internally within the Apricot F1, the Sony OA-D32W. The drive is characterised by incorporating two read/write heads and utilising 80 track double-sided MicroFloppy disks. It is possible to use (format, read from/write to) 70 track MicroFloppy disks within the 80 track drive. (The drives are physically compatible with 70 track MicroFloppies and the BIOS software is configured to support this feature).

The disk drive is mounted directly onto a metal sub-chassis assembly in the base of the System Unit. It is secured in position by four screws which connect the drive to the sub-chassis frame.

Connections from the disk drive controller, the Floppy Disk Controller on the System Board, are linked to the drive via a 26-way ribbon cable assembly. Power supply connections to the drive also originate from the System Board and are provided by a 4-wire cable assembly.

Note:

To avoid the possibility of damage occurring to the read write heads of the drives within the computer during transit, packing disks should always be inserted into the drive slots prior to transportation. If packing disks are not installed prior to transportation, excessive vibration during transit may cause the heads to crash together resulting in unrepairable damage.

Do not:

1. *Switch the machine on with the packing disk in the drive slots.*
2. *Insert the packing disks into the slots whilst the power supplies are present.*

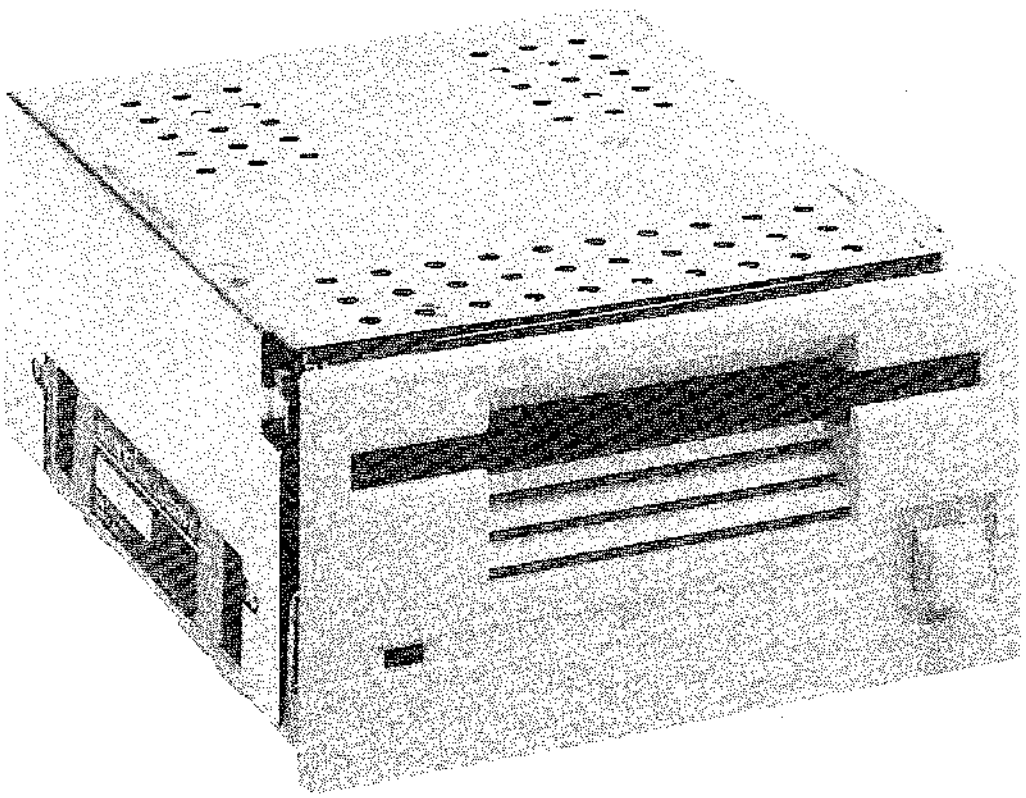


Figure 1. MicroFloppy Disk Drive

Details

General

The disk drive contains all the necessary electronics and mechanics for transferring MFM encoded serial data between the MicroFloppy disks and the System Board.

The electronics consist of:

1. The interface to the disk controller (housed on a single printed circuit control board located at the base of the drive).
2. A series of sensors for detecting various conditions within the drive (e.g. When the heads are positioned over the first track defined as Track 0 of the disk, when a disk is in the drive, etc.).
3. The read/write head transducer circuitry for reading and writing data from/to the disk.

The mechanics consist of the disk drive mechanism, the disk loading/eject mechanism, and the mechanisms for positioning and engaging the read/write heads.

Interface Details

Connections between the disk drive and the System Board consist of four types; MFM encoded data signals, control input signals, status output signals and power supply lines. The latter is supplied via the four wire cable assembly; the remaining three types via the 26-way ribbon cable.

All signals supplied via the 26-way ribbon cable are driven by open collector 74 series logic gates, apart from the Index Pulse, which is driven by an open collector transistor.

The function of each connection is detailed in tabular format following the interface connector location diagram.

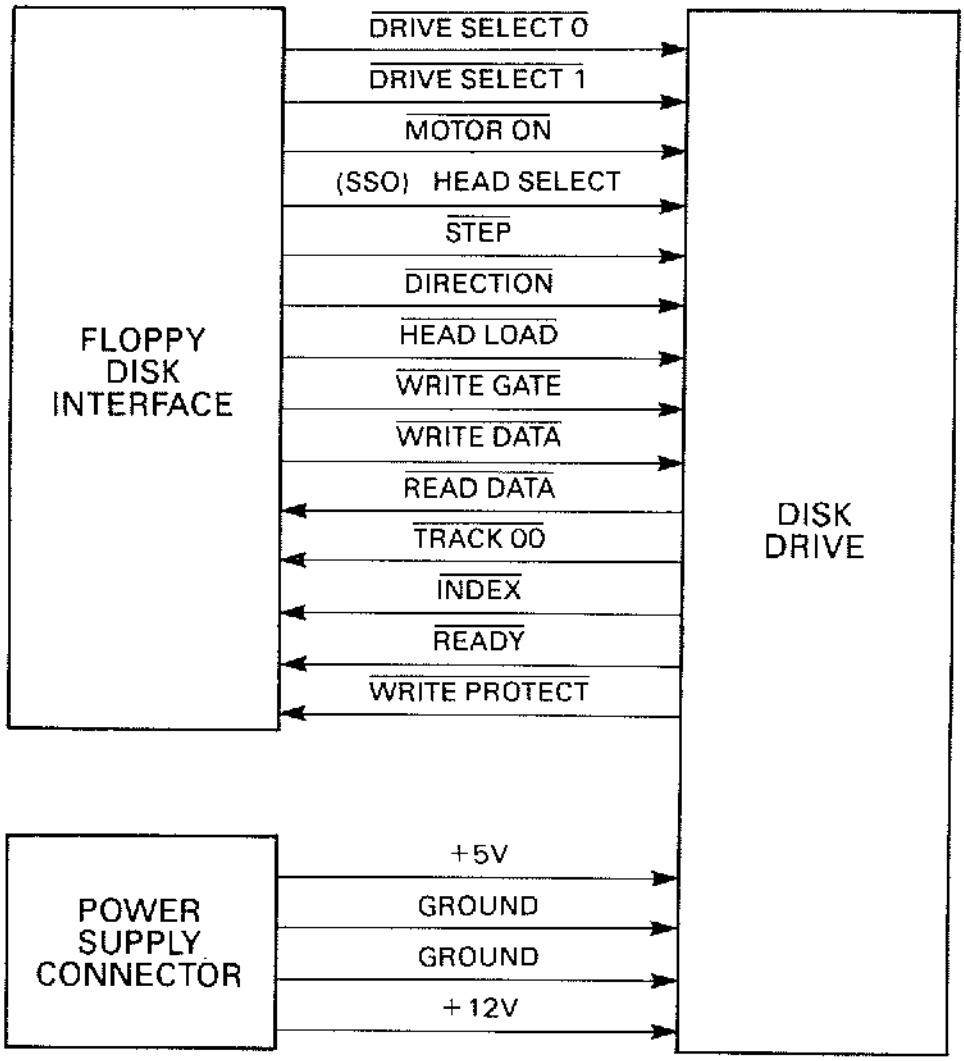


Figure 2. Interface block diagram

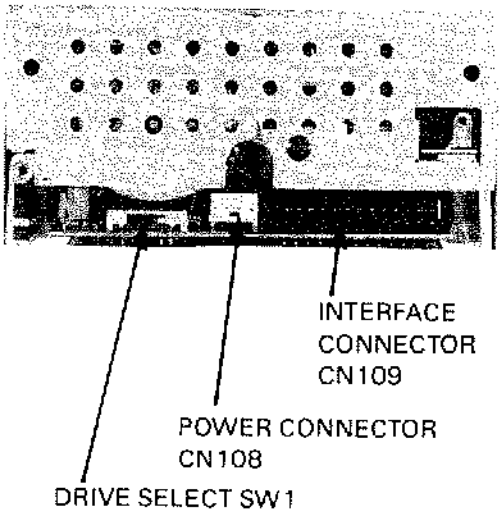


Figure 3. Connector Location

Interface Connections (Outputs)

$\overline{\text{Ready}}$	Active state, logic low. When active, indicates that a disk is within the drive, the drive is selected, and the drive motor is rotating at the normal operational speed (i.e. the drive is available for a data transfer operation). The time taken for $\overline{\text{Ready}}$ to be set active with two other conditions already met following: 1. Drive select being set active is 0.5 μs. 2. Insertion of a disk is of the order of 1.6 seconds. 3. Motor on set active is in the order of 400 ms.
$\overline{\text{Write}}$ Protect	Active state, logic low. When active, indicates that the disk is write protected. If the drive is not selected, the output is set to logic high, regardless of the disk status.
$\overline{\text{Index}}$	Index pulse which acts as the reference for the start of a track. Short duration negative going pulse (150 to 350 μs), generated once per revolution of the disk (i.e. every 100 ms at normal operational speed). If the drive is not selected, the output is held at logic high.
$\overline{\text{Track 00}}$	Active state, logic low. When active, indicates that the heads are positioned over the first track on the disk. If the drive is not selected, the output is set to logic high regardless of the position of the head.
Read Data	MFM encoded serial data stream from the disk during disk reads. If the drive is not selected, the output is forced to logic high.

Interface Connections (Inputs)

Drive Selects	Drive Select 0 and Drive Select 1. These outputs provide the capability for selecting one of the drives in a dual disk drive environment. The combination of states on these lines determine which drive is selected for operation. Normal decoded select status from the System Board is one Drive Select set low with the second select line set high. If Drive Select 0 is set low, the drive configured as drive 2 by a switch at the rear of the unit is selected for operation (see Drive Switch Settings for more details). Conversely, if Drive Select 1 is set to logic low, the drive configured as drive 3 is selected for operation.
Motor On	Active state, logic low. Controlled by the BIOS software. The drive is configured by a switch on the underside of the unit so that the drive motor is switched on only when a disk is within the drive and Motor On is active.
Step	A negative going pulse generated by the disk drive controller which moves the read/write heads, if the drive is selected. Each pulse causes the heads to be moved to an adjacent track location, in the direction specified by the direction input.
Direction	For each valid step pulse, the head moves in one track location towards track 79, if the direction control line is at logic low; and one track location towards track 0, if at logic high. If the head is already at either track 0 or track 79 and a step pulse is issued with the direction input set to move the head outside the normal track range, the head is held stationary.

Contd...

Head Select	Control signal used to select the side of disk for the data transfer operation (Side Select output from the disk controller). A logic low sets the lower read/write head into the active state enabling data to be transferred to/from Side 0 of the disk (providing Head Load is active). Conversely, a logic high sets the upper read/write head active enabling data to be transferred to/from Side 1 of the disk (providing Head Load is active). The time taken for a read/write head to become ready for data transfers, following a change of state of the Head Select signal is 100 μs.
$\overline{\text{Head}}$ Load	Active state, logic low. When active, causes the read/write heads to make contact with the disk surfaces, providing the drive is selected. If the drive is deselected, whilst the head load signal is still active, the head remains loaded.
$\overline{\text{Write}}$ Gate	Active state, logic low. When active; enables the drive write control circuits to receive the write data from the disk controller; switches current through to the read/write head selected by the Head Select signal; and also enables the selected head's tunnel erase head. Set to the inactive high state during disk read and all head positioning operations.
Write Data	MFM encoded serial data. Changes the polarity of the current flowing through the selected read/write head on each negative-going transition, providing the following conditions are met: 1. The drive is selected. 2. The Write Gate input is active. 3. A write unprotected disk is inserted. 4. The drive motor is rotating at operational speed. 5. The heads have been loaded and are stationary.

Disk Drive Mechanism

The disk drive mechanism is a brushless direct drive motor, which employs a velocity servo control circuit to ensure that the disk rotates at a constant speed of 600 rpm. The drive is configured so that the motor rotates only when a disk is within the unit. Removal of the disk from the drive causes the motor to stop. The time taken for the motor to reach the normal operating speed following the insertion of a disk, is the order of 1.6 seconds.

The servo control circuit also generates the index pulse once per revolution of the disk.

Read/Write Heads

The two heads each consist of; a read/write element and a pair of tunnel erase heads, and are mounted opposite each other on the head guide arm. The tunnel erase section provides guard bands for adjacent tracks. Current is supplied to the read/write element of the selected head, on receipt of an active Write Gate signal from the disk controller which also activates the tunnel erase section.

Head Positioning Mechanism

The head positioning mechanism uses a stepping motor and a guide arm controlled by a needle screw to precisely position the read/write heads over the tracks on the disk. Control of the movement of the heads is supplied by the Step and Direction inputs from the disk controller. On application of the power supplies, the drive automatically generates control signals to position the heads over Track 0.

Head Load Mechanism

Head loading is controlled by the Head Load signal from the System Board. When the signal is active, one head makes physical contact with the lower surface of the disk; the second head makes physical contact with the upper surface. The Disk indicator on the front panel of the unit remains illuminated, as long as the head remains loaded. Head selection is determined by the Head Select input. This selects the side of the disk for a transfer operation by activating the read/write head transducer circuits.

Sensors and Detectors

A series of photo-sensors and associated detector circuits are fitted in the drive. These generate status output signals to the disk controller, on detecting the conditions detailed below.

1. A disk is within the drive (Ready).
2. The disk is write protected (Write Protect).
3. The heads are positioned over Track 0 (Track 00).
4. The start of each track (Index Pulse).

Drive Switch Settings

Two switches are located on the drive, which are set according to the application of the drive within the system. One switch (SW S101) determines which of the two drive select input signals switch the drive to an operational condition. The second switch (SW S102) determines the method of switching on the disk drive motor.

The drive select switch is a 4-position switch located at the rear of the unit, as illustrated below.

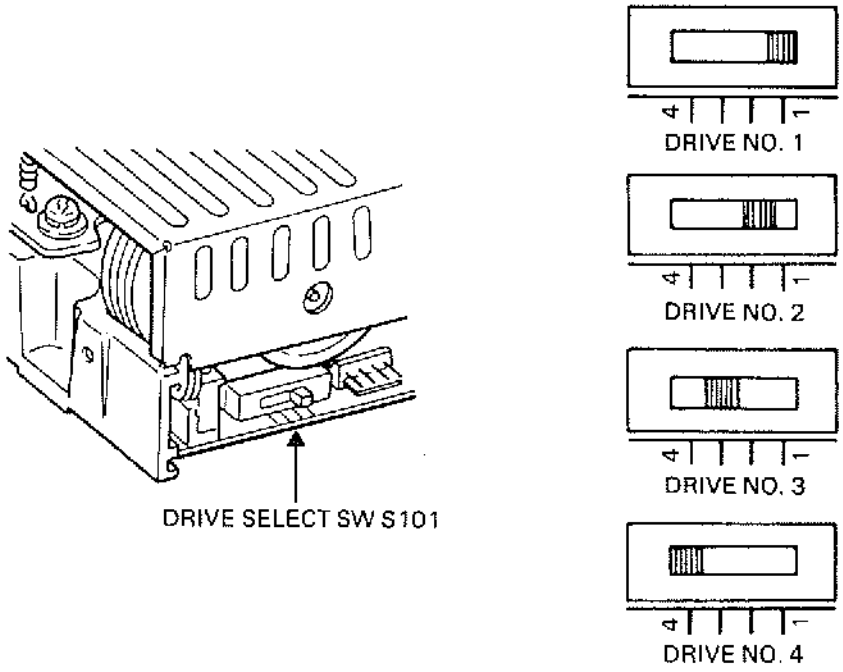


Figure 4. Drive Select Control

This switch (in theory) provides for four drives to be daisy-chained, with each one able to be uniquely selected by the two drive select input lines.

The decoding circuit of the Floppy Disk Interface circuitry on the Interface Board only allows two different combinations of states on the drive select input lines. These are drive select 0 at logic low, with drive select 1 at logic high; and drive select 0 at logic high, with drive select 1 at logic low. Due to this fixed decoding on the drive select inputs, the switch has to be set to the positions as detailed in the following paragraph.

In the standard single disk drive system, the switch on the internal drive is normally set to drive position 2. If an external second drive is daisy-chained with the internal disk drive, this is normally configured as Drive 3.

The motor control switch is located on the circuit board in the base of the unit, just behind the front panel (see illustration below). The switch must be set to position B, so the disk motor rotates only when a disk is within the drive and the contol input Motor On is active. Setting the switch to position A, will cause the disk motor to rotate, regardless of whether a disk is within the unit or not, when Motor On is set active.

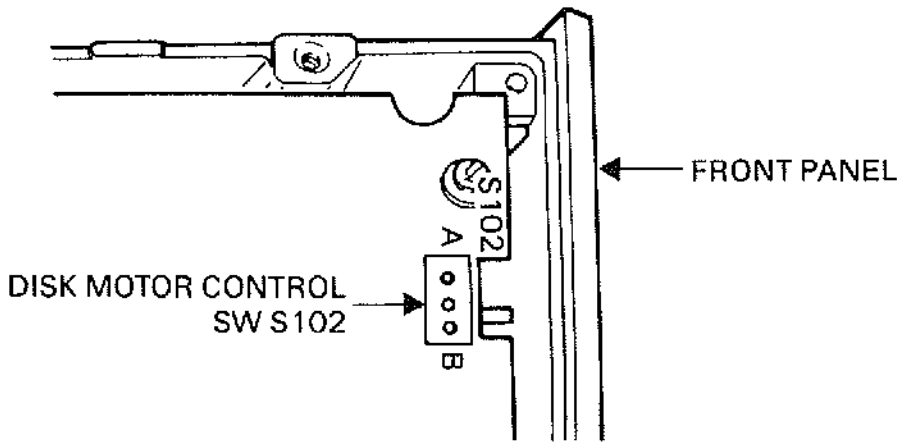


Figure 5. Drive Motor Control

Drive Specification

Media	3.5 inch 80 track double-sided MicroFloppy Disks. Soft-sectored with 9 sectors per track, 512 bytes per sector, producing a total formatted data capacity of 720 Kbytes per disk. Encoded with double density MFM data.
Data transfer rate	500 kbits/s.
Media life	More than 3×10^6 passes/track.
Track density	135 tracks per inch.
Track-to-track access time	12 ms.
Head load time	60 ms.
Head settling time	30 ms.
Head select time	100 μs. (Time taken for a read/write head to become operational, following a change of state on the Head Select input.)
Disk rotational speed	600 rpm (power-up time approx. 1.6 seconds after insertion of the disk).
Dimensions:	
Height	2.0 inches (51mm).
Width	4.0 inches (102mm).
Depth	5.1 inches (130mm).
Weight	1.5 lbs. (700g).
Current consumption	+12V supply: Typically 0.3A (max. 1.5A) +5V supply: Typically 0.48A (max. 0.8A)
Environmental conditions:	
1. Operating	
Temperature	40F to 115F (5C to 45C).
Humidity	20% to 80% relative humidity, with a wet bulb temperature of 85F (29C) and no condensation.
2. Storage	
Temperature	-40F to 140F (-40C to 60C)
Humidity	5% to 95% relative humidity,(no condensation).

Disks

General

Note: *To ensure complete compatibility with the Apricot, only ACT approved MicroFloppy disks should be used with the disk drive. Using non-ACT disks, not manufactured to the same high standard, may result in intermittent read and write errors, rendering information stored on the disk totally unintelligible.*

The Sony OA-D32W disk drive are designed to use 3.5 inch 80 track double-sided MicroFloppy disks. The disks are encased in a rigid plastic shell and feature an automatic shutter and a metal centering hub.

The shutter protects the disk media from contamination by dust, dirt or fingerprints, allowing the disk to be easily handled without affecting the integrity of stored data. When inserted into the drive, the shutter automatically slides open, allowing the read/write head of the drive access to the recording media. Removal from the drive automatically closes the shutter.

The metal centering hub ensures that the disk is accurately positioned on the disk drive motor spindle.

Disk Precautions

The same precautions apply to the MicroFloppy disks as any other magnetic recording media.

Do not:

1. Touch the disk surface.
2. Allow the disk to be placed in the proximity of other magnetic materials or other sources of magnetic fields.
3. Expose the disks to heat or direct sunlight.
4. Attempt to clean the disk surface.

Disk Insertion/Removal

Insert the disk into the drive shutter side first, with the metal centering hub facing downwards. The disk should slide into the drive with the minimum degree of force. Immediately the disk is inserted, the heads are momentarily loaded to seat the disk properly onto the drive spindle. Improper insertion results in the disk not being accepted by the drive.

Remove the disk by pressing the disk eject button. The disk eject button should not be pressed, whilst the Disk indicator is illuminated.

Write Protecting

Write protecting the disk is achieved by sliding the tab (as illustrated below) to the lower position, creating a window in the lower left corner of the disk. To allow the disk to be written to once more, slide the tab back over the window.

Disk Format

Each of the 160 tracks on the disk is divided into sectors under software control (i.e. soft sectored), with the beginning of each track indicated by the index pulse generated by the drive motor assembly. The software track format chosen for the disks is a derivation of the IBM system 34 format for 8 inch disks. This uses double density MFM encoded data, with 9 sectors per track and 512 bytes per sector. The total storage capacity of the 80 track double-sided disk is thus 720 Kbytes of formatted data (160 x 9 x 512 bytes).

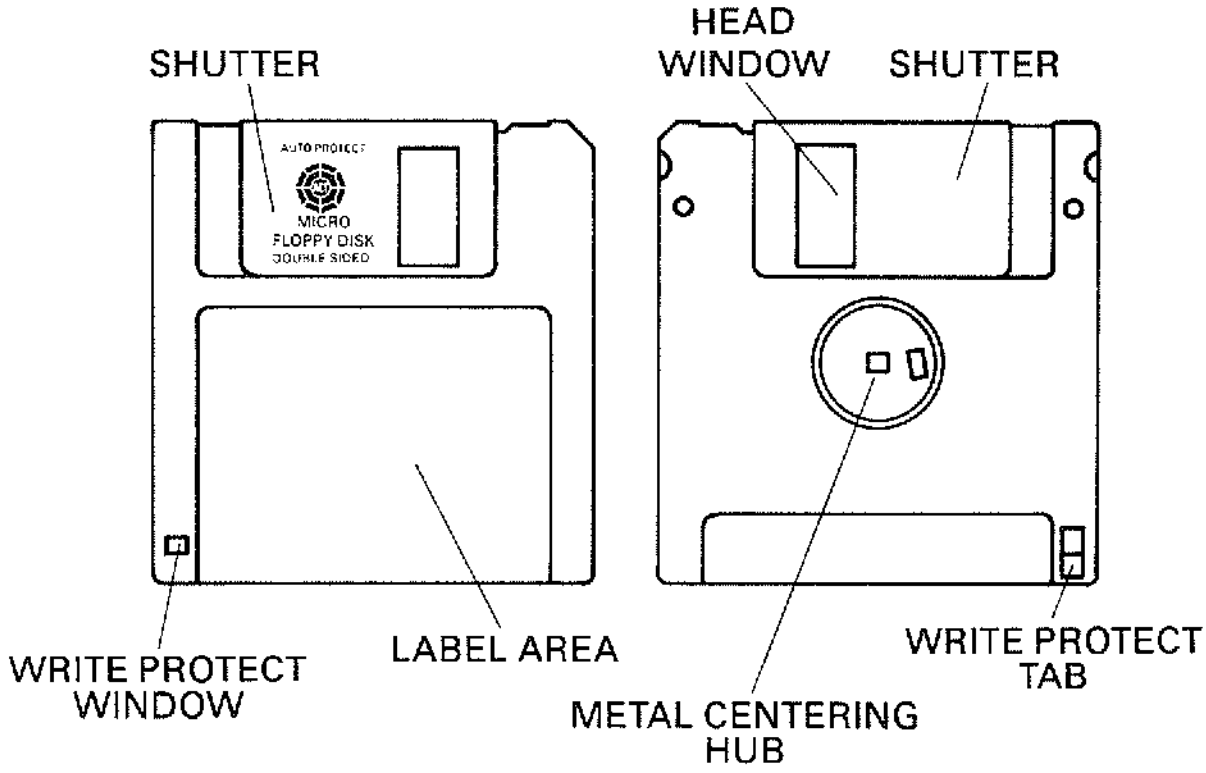


Figure 6. Microfloppy Disks

Keyboard 2.12

Contents

Introduction

Details
- Mechanics
- Circuitry
- Keyboard Scanning
- Data Transmission Format
- Keycode Data Encoding
- Special Keys

Illustrations

1. Keycode Transmission Format
2. Synchronous Packet Format
3. X-Y coordinates

Introduction

The Keyboard for the Apricot F1 is a full function keyboard featuring; 92 keytops, four "special" keys, a time and date clock implemented in software, and an infra-red interface for transmission of keycodes and data to the Systems Unit.

It is an identical design to the Keyboard found on the Apricot Portable and is directly compatible and interchangeable. It employs the same circuitry, the same key layout, and uses the same keycode encoding scheme/data transmission rate as found on the Portable keyboard. (The only difference is in the colour of the Keyboard plastics).

The maximum practical distance of operation is specified at 2 metres. The Keyboard will work at greater distances than this but the difficulties in viewing the screen so far away in any other mode apart from 40 column render this sort of usage of academic importance only.

A "light-pipe" is supplied with the F1 to connect the Keyboard and Systems Unit together in multiple machine environments where interference from other users may occur. One end of the pipe plugs into the recessed LED socket on the front edge of the Keyboard. The other end must be plugged into the right hand LED socket on the Systems Unit (as viewed from the front) to prevent interference from other infra-red sources.

The power supply for the Keyboard is provided by four AA cells. These are located behind a cover panel accessible from the underside of the Keyboard. The operational lifetime of the batteries is approximately 6 months under normal everyday usage.

Details

Mechanics

The mechanics of the Keyboard are of a comparatively simple design, consisting of a single circuit board, the main key switch array, the four fixed function keys and four AA batteries.

The batteries are located behind a removable cover panel in the base of the Unit. A coin or flat bladed tool is required to gain access to the battery compartment.

The Keyboard PCB is also fitted behind a removable cover panel. As user access is not necessary to this area, there is no easy coin-release slot.

The main key switch array is a single component which is clipped into the plastic moulding and cannot be removed once in position without damaging the array. It is linked to the circuit board by a multi-way ribbon cable connector.

The four fixed function keys are tracked directly onto the keyboard PCB.

The key tops of the key switch array are removable and can be easily repositioned for custom keyboard designs. Applying slight leverage underneath a key top releases it from its normal location, but care should be taken not to lose the springs.

Three infra-red LEDs are mounted on the Keyboard PCB and are located on the front edge of the Keyboard. The LEDs are spaced across the width of the Keyboard to provide a decent spread of infra-red signal. This removes any restriction on having to place the Keyboard directly in front of the Systems Unit, and thus allows the user the maximum degree of flexibility in siting the Keyboard.

Two of the LEDs are slightly proud of the front edge of the Keyboard, the third is recessed. The third LED is recessed to allow the light-pipe to be plugged in. Transmissions from the Keyboard are then supplied directly to Systems Unit down the fibre optic link.

The light-pipe does not switch off transmissions from the other two LEDs; these will continue to transmit infra-red when a key is pressed even though the light-pipe is in position. (Connecting the light-pipe into the right hand socket on the front of the Systems Unit switches off the infra-red receiver surrounded by the wide-angle lens, so that infra-red from other sources will not be detected - See Systems Unit chapter for more detail).

Two buttons are located on the sides of the Keyboard (one on each side). These release the spring-loaded feet which tilt the Keyboard for normal desk-top usage.

Circuitry

The circuitry on the Keyboard PCB consists of an NEC 7507 microprocessor, three infra-red LEDs and assorted components for interfacing the 7507 processor to the keyswitch array and LEDs. A circuit diagram of the Keyboard is provided in an appendix to this manual.

The 7507 is a CMOS 4-bit single chip microcomputer which has its own internal ROM and RAM. It also contains an internal timer, a vectored interrupt structure and features 32 I/O lines. The I/O lines are organised into eight 4 bit ports. The ports are identified by a prefix which is the port number followed by the port line number. e.g. P21 is port 2, line 1).

Keyboard Scanning

The 7507 uses the standard method of row/column scanning for detecting key closures in the keyswitch array and also the closure of the four fixed function keys. The four fixed function keys are mapped onto the bottom row of the keyswitch matrix (port P60).

The scanning method is as follows. The 7507 selects a row by setting the appropriate port output to logic low. It then sequentially scans through all the columns looking for a logic low input on the appropriate input ports which indicates a keyclosure.

The Shift and Control keys are not scanned in the same way and are treated as special keys. These are wired directly to input ports.

The keyboard processor is normally in a sleep mode and is awakened every 15.6 ms. The 15.6 ms timer routine performs two functions. It checks for key closures and also updates the software time/date clock.

If a key has been pressed, the keyboard remains awake and decodes the selected key, formats it, and then transmits it to the Systems Unit by pulsing the infra-red LEDs. If no other key has been pressed the keyboard re-enters sleep mode to conserve battery power.

When more than one key closure is detected in a single scan, the 7507 software checks the validity of the closures to prevent false key codes being issued. It does this by reducing the area to scan by searching a "closed" key switch map until it finds an unambiguous key closure.

Only keys which can be uniquely identified are encoded and transmitted.

The total time taken between detecting a key closure to the end of the keycode transmission is normally between 26 to 32 ms. At the end of the transmission, the processor rescans the key switch and then enters sleep mode if no further key is active.

The oscillator connected to the X1, X2 inputs of the processor sets the sleep mode cycle. The processor cycle time is set by the components connected to CL1/CL2.

Data Transmission Format

The majority of the consumption of power on the keyboard is taken by the three LEDs. In order to extend the battery life, these are normally switched off and only pulsed on for the transmission of data.

All keycode data is encoded into serial packet format, consisting of 32 bits, prior to transmission (see below for more details on the actual encoding format used). The 32 bit serial data stream is sent via the data output (P31) to drive the three LEDs.

The data is converted into a pulsed waveform by a monostable circuit prior to forming the drive signal for switching the LEDs on. The duration of the pulses and thus the time the LEDs are switched on, is kept relatively short (of the order of 15 μs) to minimise the amount of battery power consumed.

The monostable circuit translates the bit stream into the waveform format as shown in Figure 1. This is in effect a transmission packet consisting of an interleaved clock and data waveform. Encoding the data in this way allows the decoding circuits on the Systems Unit to easily compensate for variations in the data rate.

Data "1s" are signified by a pulse within the clock pulse time period and data "0s" signified by an empty time slot.

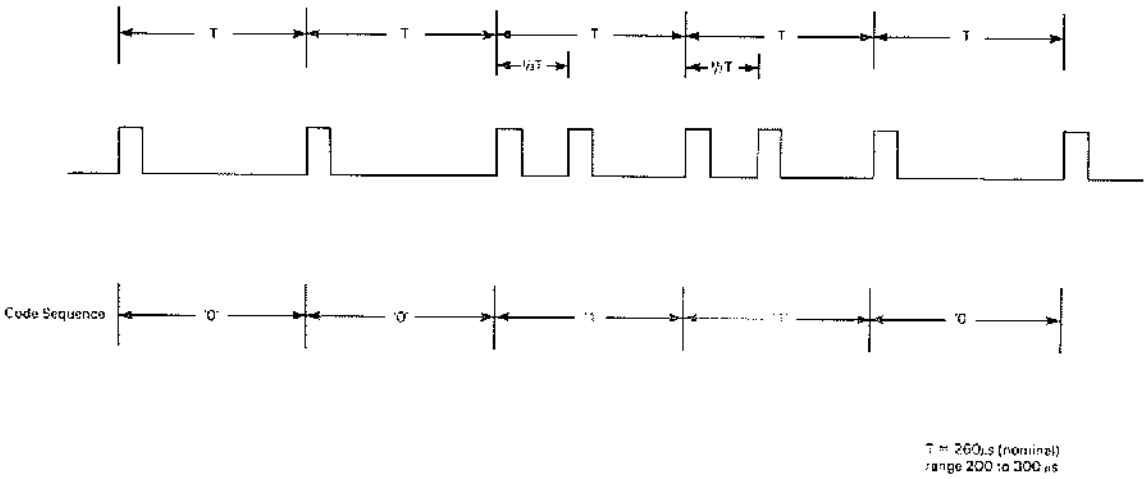


Figure 1.Keycode Transmission Format

Keycode Data Encoding

To maximise system reliabilty, and ensure that data transmitted from the Keyboard is not misinterpreted on the Systems Unit, the keycode data is encoded into a special format.

The Keyboard formats the valid key closures into a serial packet consisting of a four byte sequence. The format is the synchronous transmission mode Monosync and is operated at a fixed data rate of approximately 3.85 Kbits/sec. The first byte is the sync header. All the following bytes are the actual data. These are encoded using Hamming codes (see Figure 2).

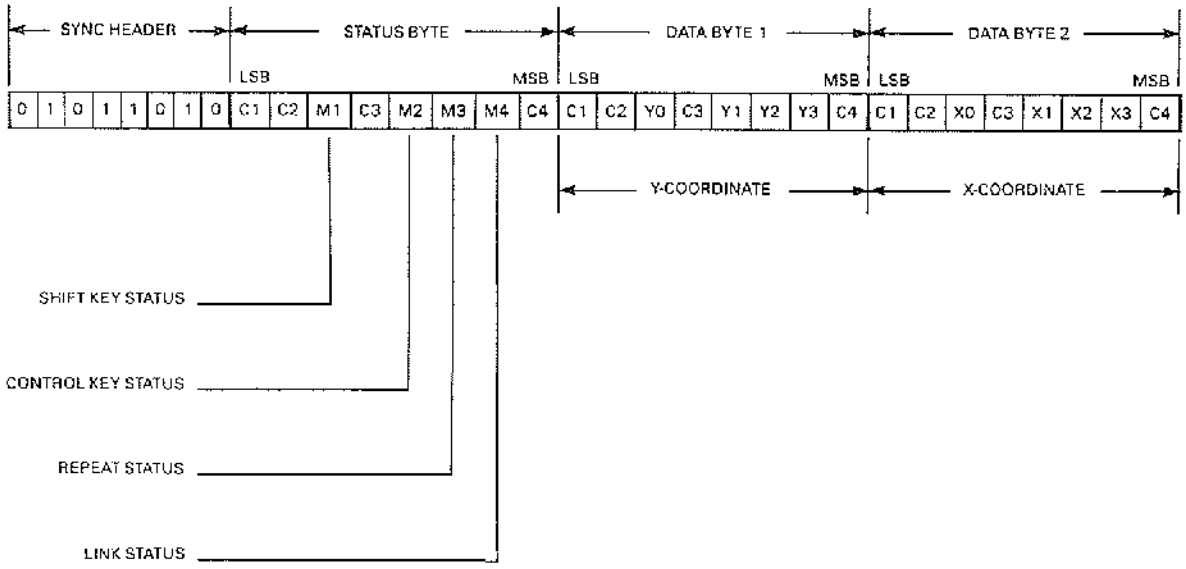


Figure 2. Synchronous Packet Format

This method enhances system reliability in two ways:

1. Using Monosync means that the receiver circuits on the Systems Unit (the Z80 SIO of the serial interface), have to first detect a valid data pattern, (the sync header byte) before it regards the data sent as being valid. This provides a high degree of protection from other infra-red sources, as they will not contain the sync header and will therefore be totally disregarded.
2. Using a Hamming format to encode the data enables the BIOS software in the Systems Unit to check the integrity of the data received from the Keyboard. It produces a highly reliable system for proving the validity of the Keyboard data, providing a measure of protection against a transmission which contains a valid sync byte, but invalid data (missing or corrupted data).

The four byte sequence consists of the Sync header byte (5AH), a status byte and two data bytes. The status byte and keycode data bytes are encoded with a Hamming format.

The status byte contains four bits (a nibble) of information as follows:

bit 0 Whether the key pressed is pressed in conjunction with the SHIFT key. This state is indicated when the Shift bit is set high.

bit 1 Whether the key pressed is pressed in conjunction with the CONTROL key. This state is indicated when the Control bit is set high.

bit 2 Whether the key code transmitted was the same key code as transmitted immediately prior to the current transmission. The Repeat Status bit is set high to indicate that a key is being held down (repeated).

bit 3 That the data transmitted to the Systems Unit is from the Keyboard. This bit is indicated by the Link Status bit and is always set low for a keyboard transmission. (This bit allows the BIOS to differentiate between Keyboard and Mouse data which uses a similar format but sets the Link Status bit high).

The first keycode data byte contains a data nibble (Nibble 1) which is the Y-coordinate of the selected key in the keyswitch matrix. The second data byte contains a data nibble (Nibble 2) which is the X-coordinate of the selected key in the keyswitch matrix.

The X-Y coordinates for each key are illustrated on Figure 3. The coordinates are identified by the numbers in the top left-hand corner of each key. For example, the CAPS LOCK key is located at X-coordinate 2H, Y-coordinate BH (i.e. the keyswitch matrix position 2,11 in decimal format).

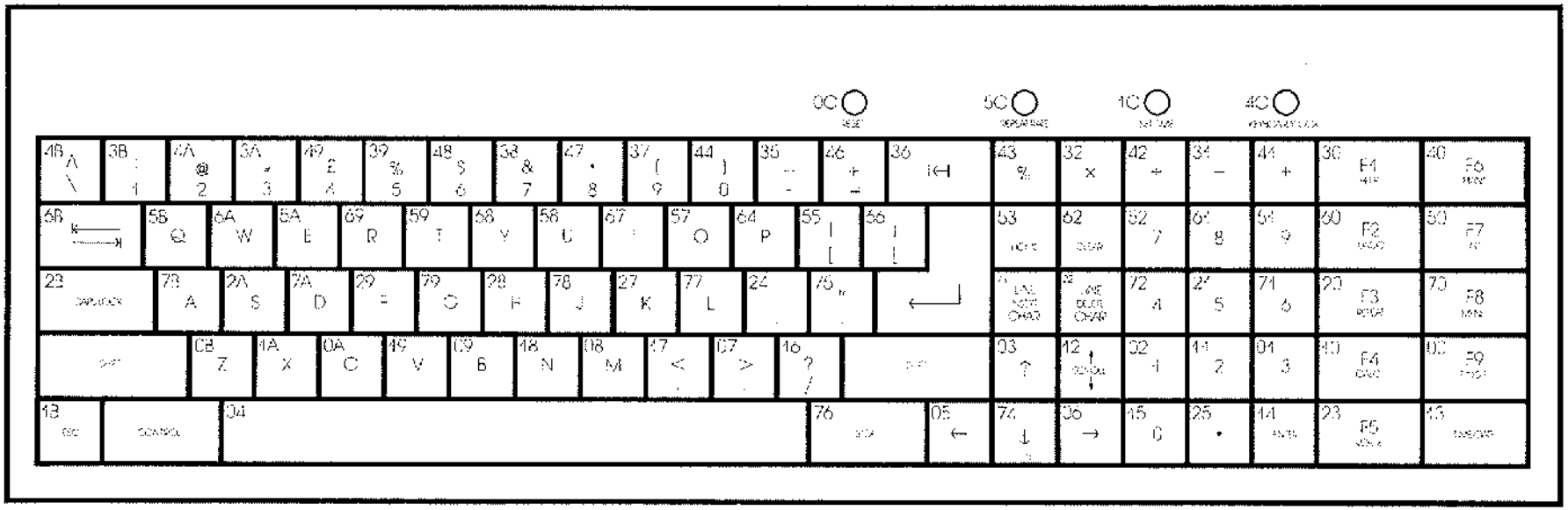


Figure 3. X-Y Coordinates

All keys apart from the RESET, REPEAT RATE, SET TIME, TIME/DATE (and SHIFT and CONTROL which are not in the matrix) are transmitted from the keyboard by encoding the X-Y position of the detected key with Hamming codes in the format as described above and perform no other function.

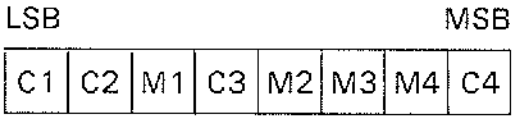
The handling of the RESET, REPEAT RATE, SET TIME and TIME/DATE keys are slighly different and are discussed under the section headed Special Keys. The SHIFT and CONTROL keys are encoded into the status byte only and are never transmitted as a X-Y keycode.

Hamming Format

The encoding of the status and keycode data nibbles uses a Hamming distance of four. (The definition of the Hamming distance equates to the fact that for all possible combinations of encoded bytes, each byte will have at least four bit positions different when compared with all other encoded bytes).

A Hamming distance of four allows the BIOS to detect any two-bit errors and also correct any single-bit errors in the transmitted data. Each data nibble is converted into a Hamming encoded byte by the addition of four check bits.

The Hamming encoded bytes are produced by generating the check bits and then combining them with the nibble of data in the format detailed below.



Hamming encoded Byte

C* corresponds to a check bit

M* is a bit within the data nibble. The position of the LSB is specified by M1, the MSB by M4.

The check bits are generated using the algorithms detailed below.

$$C1 = M1 \oplus M2 \oplus M3$$

$$C2 = M1 \oplus M3 \oplus M4$$

$$C3 = M2 \oplus M3 \oplus M4$$

$$\overline{C4} = C1 \oplus C2 \oplus M1 \oplus C3 \oplus M2 \oplus M3 \oplus M4$$

Since a nibble of data can only have one of 16 values (0H to FH), the data nibbles can be translated into the corresponding Hamming encoded bytes by using a simple look-up table. This is as follows:

Data Nibble	Encoded Byte
0H	80H
1H	07H
2H	19H
3H	9EH
4H	2AH
5H	ADH
6H	B3H
7H	34H
8H	CBH
9H	4CH
AH	52H
BH	D5H
CH	61H
DH	E6H
EH	F8H
FH	7FH

Special Keys

Some of the keys are not transmitted from the Keyboard in the same way as decribed above. These are the four keys RESET, REPEAT RATE, SET TIME and TIME/DATE.

Reset

When the 7507 microprocessor detects that the RESET key has been pressed it does not transmit a hamming encoded keycode. Instead, it sends sync characters (the byte value 5AH) at a set rate of one sync character every 15.6 ms for as long as the key is held down.

If the key is held down for approximately one second, the accumulative effect of receiving the multiple sync bytes on the Systems Unit generates a hardware reset. This mechanism is discussed in the System Detail chapter.

Repeat Rate

The Repeat Rate key is never transmitted to the Systems Unit. Instead it has the effect of acting as a toggle for varying the rate at which characters are sent from the Keyboard when a key is held down (i.e. the auto-repeat rate). It allows the rate of transmission to be either one of two values:

1. Eight characters per second which is the slow auto-repeat rate setting (one transmission every 125 ms).
2. Eighteen characters per second which is the fast auto-repeat rate setting (one transmission every 55 ms).

Set Time

The function of the SET TIME key is to allow the user to reset the time and date of the internal keyboard clock.

When the SET TIME key is pressed, the 7507 stops the internal clock and transmits the SET TIME keycode to the Systems Unit. The ROM BIOS on the Systems Unit decodes this key and activates the 25th line on the display to prompt the user to type in a new time and date. This action by the BIOS happens both before and after boot.

The user must then type in the time and date as a string of digits using the numeric keypad. Each digit is transferred to the Systems Unit in the normal four-byte Hamming encoded format. Any non-numeric key during the programming sequence causes the 7507 to restart the internal clock and transmit the non-numeric keycode. This informs the BIOS to terminate the operation.

The user must type in the sequence in the order detailed below with no spaces or other character between the digits (e.g. 1000011285 sets the clock for 10 am on the 1st December 1985).

HHMMDDMMYY

At the end of the 10 character sequence, the user must then press any key to set the internal clock to the specified time and date and also restart it. (Seconds are automatically set to zero by this action).

The updated time and date information is not used by the BIOS unless the information is re-transmitted by the Keyboard. This action is performed by the TIME/DATE key.

Time/Date

The function of the TIME/DATE key is primarily to transfer the time and date generated by the internal Keyboard clock to the ROM BIOS clock. It also performs another function, that of transmitting the version number of the Keyboard software.

When the key is pressed, the 7507 transmits a sequence of 16 data packets to the Systems Unit. Each packet is in the usual four byte synchronous format and are separated by 35 ms intervals. The first data packet is the TIME/DATE keycode packet. The following packets include the software version number and the time and date information.

The format of the sixteen data packet sequence is as follows:

Packet	Pre-Hamming Code Format*	Function
1	S031	Time/Date keycode
2	S41x	Revision number
3	S42x	Hours (tens)
4	S43x	Hours (units)
5	S44x	Minutes (tens)
6	S45x	Minutes units)
7	S46x	Seconds (tens)
8	S47x	Seconds (units)
9	S48x	Day (tens)
10	S49x	Day (units)
11	S4Ax	Month (tens)
12	S4Bx	Month (units)
13	S4Cx	Year (tens)
14	S4Dx	Year (units)
15	S4Ex	Not currently used
16	S4Fx	Not currently used

* The first value (S) is the Sync byte.

The second value is the Status nibble. The value of 4 in all the packets following the keycode corresponds to the Repeat Status bit set.

The third value in all packets apart from the Time/Date packet is a data packet identity nibble.

The fourth value (x) corresponds to the data nibble as described in the Function Column.

Section 3
Software
Detail

Guide to the BIOS 3.1

Contents

Introduction

Bootstrap

Initialised drivers

Initialised BIOS

Built-in functions

Memory Map
- Software interrupts
- Hardware interrupts
- Pointers
- Ascii and Bit Screen Images
- RAM BIOS for MS-DOS

Disk Label Sector and Configuration Table

Illustrations

1. Memory Map

Introduction

The BIOS is the Basic Input Output System of the Apricot. It exists as two distinct parts in the machine, a permanent part in ROM and a loaded part in RAM.

The ROM BIOS provides the primitives necessary for powering up the Apricot or performing a system reset. It contains the hardware independent Drivers which control input and output to the hardware devices and diagnostics software.

The ROM BIOS also contains a Generic interface called the Control Device. This interface enables portable software to be written for the Apricot family even though Hardware Devices and their drivers within the ROM are totally different.

The RAM BIOS and other non-resident System files are loaded and initialised by the Bootstrap facilities within the ROM BIOS.

The RAM BIOS is the interface between the operating system and the ROM BIOS. It is unique to the Operating System — in this case MS-DOS.

The following sections provide a guide to the BIOS and in particular to the following:

1. The "Bootstrap" procedure
2. The Label sector, system configuration data and minimal requirements of a Boot disk
3. The initialisation of the BIOS and Device drivers
4. The Memory map and reserved areas
5. User interrupts

Bootstrap

The Bootstrap procedure is invoked by either a "cold start" i.e. powering up the machine or from a "warm start" by pressing the keyboard RESET button and holding it down for approximately 1 second.

A "cold start" performs diagnostic functions and a complete configuration of the Operating system.

A "warm start" assumes that certain data created by the "cold start" is intact but to ensure that this is reliable it consults the "water mark" data in RAM for pre-defined values. The "water mark" is detailed in a later section on Pointers. The diagnostics are not run.

Following either start up the hardware devices whose drivers are resident in the ROM BIOS and their respective data areas in the RAM BIOS are initialised.

The initialisation of each driver is discussed in the next section.

The ROM BIOS then displays the start up screen which includes a logo, total capacities of floppy (in Kbytes) and Winchester disks (in Megabytes) where applicable together with the RAM size (in Kbytes).

If a diagnostics error occurs then a failure code, as detailed in Appendix A, is displayed.

The ROM BIOS then initialises a 10 second timeout, displays "Flashing hand" and key icons and waits for the operator to press the TIME/DATE key. Depression of the TIME/DATE key before the timeout elapses ensures that the system clock is updated from the keyboard which has a battery backed clock.

On receipt of the TIME/DATE key, or on expiry of the timeout, the ROM BIOS commences searching for a Bootable disk. The Floppy drive is searched first. If it does not contain a correct disk then if there is a Winchester, loading will be carried out from it instead.

Certain fields within the Label Sector of the disk identify a Bootable disk. The Label Sector is detailed in full in a later section. The following fields must, however, contain valid Bootstrap data:

LBL_Boot_Disk	offset	0Bh	byte
LBL_winchester	offset	0Dh	byte
LBL_boot_locn	offset	1Ah	dword
LBL_boot_size	offset	1Eh	word
LBL_boot_addr	offset	20h	dword
LBL_boot_st_off	offset	24h	word
LBL_boot_st_seg	offset	26h	word
LBL_BPBsctr_sz	offset	50h	16 bytes
to			
LBL_BPBstart_sct	offset	5Eh	

Configuration Constants Area:

CNF_ver_lo — CNF_nnn	offset	80h-FFh	128 bytes

If a Winchester disk:

WINbad_sect table	offset	100h	32 words

If the disk is a valid bootable disk, the ROM BIOS will attempt to load the sectors indicated by LBL_boot_locn and LBL_boot_size from disk into RAM at the address given by LBL_boot_addr. Note that the code to be booted must be one contiguous block of sectors.

If successful the ROM BIOS loads up the internal configuration table from the boot disk label sector (128 bytes starting at CNF_ver_lo), clears interrupts and jumps to the location given by:

LBL_boot_st_seg:LBL_boot_st_off.

If, however, there is a failure, an error number (as detailed in Appendix A) is placed on the screen, and the ROM BIOS deselects the drive until a new disk is inserted.

Initialised drivers

The list of drivers given below are resident in the ROM BIOS and they are initialised by the Bootstrap procedure as follows:

Screen Driver

This clears the display and homes the cursor.

Keyboard Driver

This clears out internal keyboard queues and status indicators, and all keyboard input is ignored except for the CALCULATOR, SET TIME, KEYBOARD LOCK, and TIME/DATE facilities.

Disk Driver

This clears internal disk statuses and then tests for a second disk drive by selecting it and checking for track 0. Note that the disk drives perform an automatic RESTORE on power-up, thus activating the TRACK00 signal. These tests take about 2.5 seconds and are only performed on a power-on reset.

Printer Driver

This clears the internal (2 Kilobyte) printer buffer, and sets the data strobe output line to idle.

RS-232 Driver

This initialises the RS-232 driver to the internal configuration of 9600 baud, 8 bit per character, no parity, and 1.5 stop bits for transmission and reception.

Clock Driver

This resets the ROM BIOS clock to midnight on the 1st Jan 1980 (days = hours = mins = secs = hundredths secs = 0000.), and sets the BIOS clock running.

Winchester Driver

This checks for the presence of a Winchester controller board by testing for RAM in the Winchester buffer port. If RAM exists there the ROM BIOS assumes the presence of a 10 Megabyte Winchester. The Winchester(s), if present, are restored, and the label sector(s) read to determine the actual size of the drive(s).

Initialised BIOS

When the ROM BIOS passes control to the operating system software loaded from disk the machine is in the following state:

All 256 interrupt vectors are initialised, including all hardware and ACT defined interrupts. Unused vectors are set to point to a dummy handler in the ROM. Detection of a non-resident driver can therefore be made by checking if it's interrupt vectors point to an area above 0F8000 hex.

A list of Software interrupts and reserved interrupts is provided in a later section.

A pointer area as described in the section Memory Map is initialised.

The Keyboard Table and Character Font are in ROM, so to use a full set of characters, and to enable the reprogramming of keys, the RAM BIOS loads a new font and keyboard into its own RAM space and changes the pointers as appropriate.

The font provided in the ROM BIOS is a standard 128 character Ascii set, implemented in an 8x8 matrix. Refer to the chapter on the keyboard driver for information on the default keyboard table.

All hardware drivers are in an initialised state and can be called immediately. However, it is recommended that these drivers are initialised via the control device INIT calls (command code 0), except for the Clock Driver which should not need resetting, before the loaded operating system makes use of them.

Built-in functions

While the ROM BIOS is waiting for a boot disk the user can make full use of the following built-in facilities:

The Calculator

This appears on the 25th line of the screen, and is activated/deactivated by the CALC key (Shift + F4).

Keys used by the calculator are: Numeric Pad 0-9 and (.), ENTER, CLEAR, %, *, /, -, +, M+ (F3), M— (F8) and RECALL (F7). After boot the calculator also responds to the SEND (F9) key.

The SET TIME Key

At any time (before or after booting) the SET TIME key can be entered. This displays a prompt on the 25th line for time and date setting of the internal keyboard clock. The user must enter the date and time as a string of digits from the numeric keypad in the form HH MM DD MM YY (no spaces). The time is sent to the Main Unit by entering the TIME/DATE key. The screen prompt is removed if an invalid key is entered, more than 10 digits have been entered, or the TIME/DATE key has been pressed.

The TIME/DATE Key

This key can be pressed at any time, its function is to transmit the keyboard's internal time and date to the main unit, and is used during the boot process to start off the boot. During normal operation there is no indication of entry of this key — the ROM BIOS simply updates its internal clock with the data received.

Memory Map

The Memory Map below shows the initialised state of the Apricot after Bootstrap is complete.

The ROM Bootstrap loads the RAM BIOS Code and Data together with SYSINIT from the disk. SYSINIT is a module responsible for loading and initialising MS-DOS and it is discarded after use.

SYSINIT is then moved to a scratch area in high RAM. It then loads the operating system MS-DOS and the system files containing the Keyboard Table and character FONTs.

The illustration shows two possible states, one where the Keyboard and FONT files are not available and the other where they are.

To load a system without RAM based font and/or keyboard table, the fields CNF_FONT_sec and/or CNF_KEY_sec in the boot disk label sector must be set to zero. In this case the limited ROM keyboard table and FONT are used by the system.

The SYSINIT area is overwritten by MS-DOS.

Absolute addresses are given on the left of the diagram, space occupied in kbytes on the right

The memory map shows both ROM and RAM areas. The area between F8000H and FFFFFH is ROM, and includes the ROM BIOS code, default keyboard table and character font, and default BIOS constants.

The keyboard table is a standard version. The font is a standard 128 character Ascii set implemented in an 8x8 matrix.

The ROM BIOS constants are copied to RAM during the boot procedure, being amended there as necessary.

The area between 000000H and 005000H is used by the ROM BIOS, and its layout is fixed; it is termed 'ROM specified RAM'. It comprises the interrupt vectors, BIOS pointer area, Ascii screen images, and the ROM BIOS data, stack and configuration table.

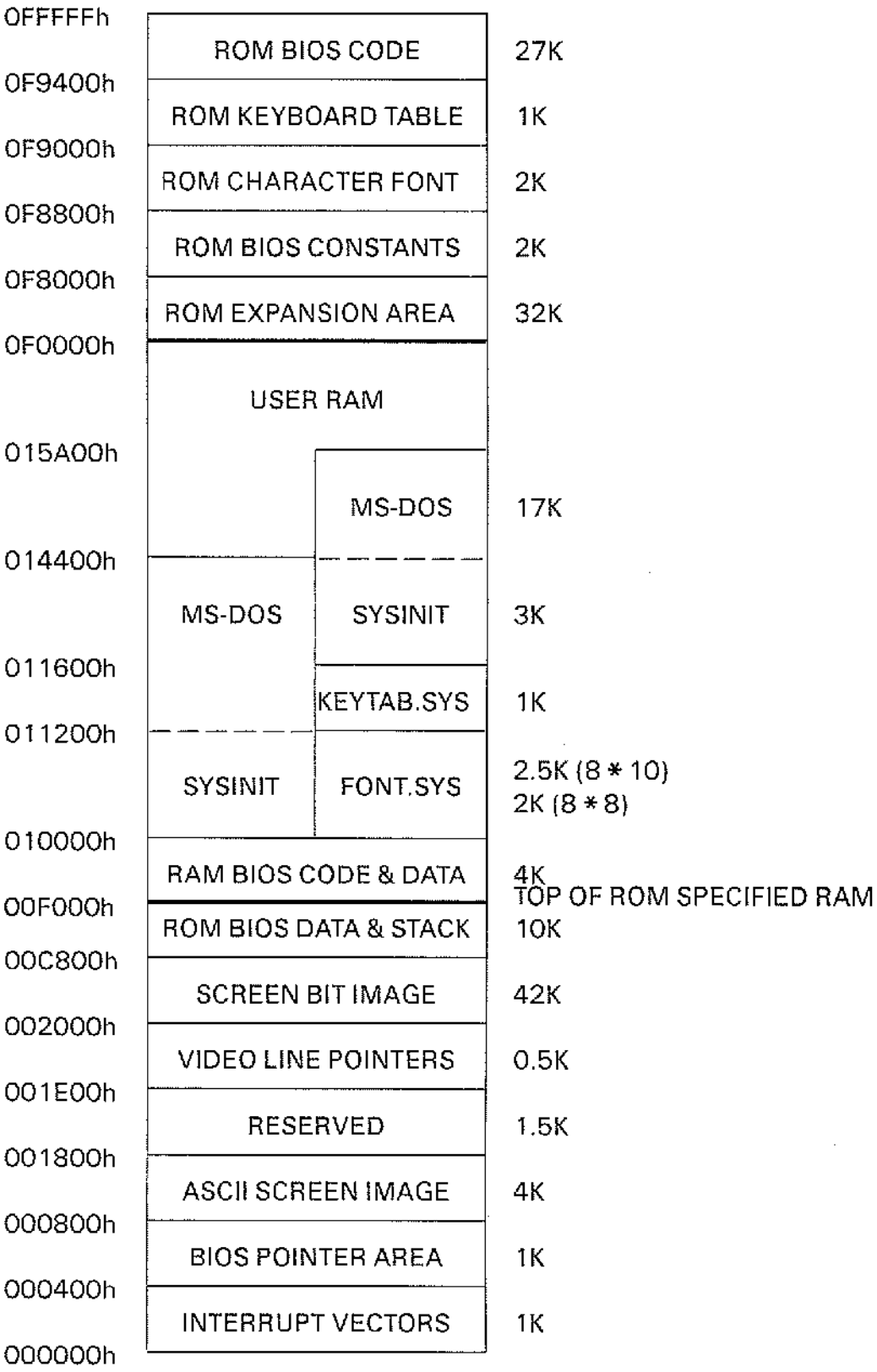


Figure 1: ***Apricot F1 Memory Map***

Software interrupts

The base page of the RAM is reserved for interrupt vectors. These are double word Segment:Offset pointers to the start address of the interrupt handler routine.

The pointers conform to Intel standard format, i.e.

<address> + 3 : segment high
<address> + 2 : segment low
<address> + 1 : offset high
<address> + 0 : offset low

The location of a pointer is determined by multiplying the interrupt number by 4; i.e. the base address of the interrupt F0 hex pointer is at location 4 x F0H or 03C0 hex.

A number of interrupts are specified as reserved by Intel and Microsoft. These are:

Intel: 0 — 1F hex
MS-DOS: 20 — 3F hex

Software interrupts are initialised to point to a dummy handler in the ROM BIOS. This handler simply executes an interrupt return (RETI).

Further interrupts are reserved by ACT for BIOS and Application use. These are listed below.

The BIOS initialises its software interrupt pointers by placing the address of appropriate routines in the respective interrupt vector location.

It is recommended that Application routines wishing to use a software interrupt observe the following rules:

1. Save the existing pointer (including the dummy pointer).
2. Store their "routine" pointer at the vector location.
3. Save all registers on entry.
4. Restore all registers before exiting the routine.
5. Perform a long jump to the "saved" pointer at the end of the routine. This ensures that any unknown routines will still be executed.

Note: The dummy routine is in ROM, i.e. all pointers to it have a segment address greater than or equal to F000 hex. This is useful for determining whether a software interrupt vector is in use. In turn this may also indicate whether a loadable device driver is present.

0FFH - Clock Interrupt every 20ms
(see note 1 below)

Source: ROM BIOS
Input: none
Output: none

0FEH - Exec interrupt

Source: Application
Input: none
Output: AX = 0FFFFH

0FDH - Spooler Interrupt

Source: Application
Input: none
Output: AX = 0FFFFH

0FCH - BIOS Control Device interrupt

Source: RAM BIOS & Application
Input: AX,BX,CX,DX,SI
Output: AX - Other registers preserved.

0FBH - Mouse Interrupt 2

Source: Application
Input: none
Output: AX = 0FFFFH

0FAH - Mouse Interrupt 1
- MicroSoft serial Mouse (see note 1)

Source: ROM BIOS
Input: AL = Mouse Data Byte
Output: Not applicable.

0F9H - Keyboard Interrupt
(see note 2)

Source: ROM BIOS
Input: AL = BL = Decoded Key
AH = BH = Count of characters in string
Output: AX = 0FFFFH - pass key to DOS queue
AX = 00000H - Ignore key
BL = Translated Character

0F8H - Voice Interrupt 3

Source: Application
Input: none
Output: none

0F7H - Voice Interrupt 2
(see note 1)

Source: ROM BIOS
Input: AX = Voice Key Code Packet
Output: none

0F6H - Voice Interrupt 1

Source: Application
Input: none
Output: none

0F5H - Mouse Interrupt 3
(see note 2)

Source: ROM BIOS
Input: AX = Mouse packet
bits 0-7 = X or Y data
bit 8 = 1 is Y packet, 0 is X packet
bit 9 = Switch 1 (1=on)
bit 10 = Switch 2 (1=on)
bit 11 = 1
bits 12-15 = undefined
Output: None

0F4H - External Expansion Interrupt Setup;
Replace External interrupt vectors and enable interrrupts.

Source: Application
Input: AL = 0 - Set External Interrupt 2 (Winchester)
AL = 1 - Set External Interrupt 3 (general)
BX = Vector Offset Word
CX = Vector Segment Word
Output: BX = Old Vector Offset Word
CX = Old Vector Segment Word

0F3H - Voice Interrupt 4

Source: Application
Input: n/a
Output: n/a

0F2H - Special 25th line output
(For calculator & specialist applications)

Source: ROM BIOS and Application
Input: AL = data
CX = command
Output: none, all registers preserved

Commands:
CX = 0, init. line 25. AL = ID code (see below)
CX = 1, reset line 25. AL = ID code (see below)
CX = 2, print character in AL
CX = 3, set cursor to position AL (0-79)

ID codes: 0=voice, 1=calc.

0F1H - Screen Output
(also provided by INT 29H)
(see note 3)

Source: RAM BIOS & Application
Input: AL = character
Output: none, all registers preserved.

0F0H - SIO Control Interrupt
(see note 1)

Source: ROM BIOS
Input: AL = SIO interrupt vector
0 = Ch B Tx buffer empty
2 = Ch B ext/status int.
4 = Ch B Rx ready
6 = Ch B Special Rx
8 = Ch A Tx buffer empty
10 = Ch A ext/status int.
12 = Ch A Rx ready
14 = Ch A Special Rx
Output: none

0EFH - Reserved for use by ACTEX
Redirected if INT 24 error handler present

0EEH - Send Non-specific end-of-interrupt (EOI) to PIC, or dummy 'RETI' sequence to Z80 chips.

Source: Application Hardware Interrupt Handler
Input: none
Output: none

0E0H - GSX-86

Source: Application
Input : CX = 0473H — Function code
DS:DX - Parameter Block pointer
Refer to GSX chapter.

Note 1: SS, DS, ES must be preserved, Stack must not be changed if interrupts are enabled, or a call to the ROM BIOS is to be made.

Note 2: All other registers must be preserved, Stack must not be changed if interrupts are enabled, or a call to the ROM BIOS is to be made.

Note 3: Interrupt 29 hex is set up by the RAM BIOS for use by MS-DOS and can be re-used by other operating systems, since screen output is already provided by interrupt F1h. The ROM BIOS only uses interrupt F1h for screen output and never interrupt 29h.

Hardware interrupts

02H - Floppy disk controller interrupt (NMI)
50H - SIO ch B TX buffer empty
52H - SIO ch B external/status
54H - SIO ch B RX ready
56H - SIO ch B special RX status
58H - SIO ch A TX buffer empty
5AH - SIO ch A external status
5CH - SIO ch A RX ready
5EH - SIO ch A special RX status
60H - CTC ch 0 expansion interrupt
62H - CTC ch 1 RS232 Baud rate
64H - CTC ch 2 sound frequency
66H - CTC ch 3 system clock (20ms)

Pointers

The ROM BIOS accesses a 1K byte Pointer area located between address 400H and 7FFH in the RAM. The pointer area is initialised by the Boot procedure. It contains contains constants and double word pointers.

The Pointers provide the BIOS and Application with details of the Operating System as follows:

1. Version number
2. Boot details and diagnostic results
3. Disk drives and their capacities
4. Table pointers and sizes
5. Memory map details

Certain pointers may be changed freely by the Applications, such as those that point to character Fonts, but others must NOT be altered.

The Table given below details each pointer, it's size in bytes and an indication of whether it is freely modifiable. All data and pointers can of course be referenced by the Application.

Double-word pointers (four bytes) conform to the standard Intel addressing formats, i.e. the first two bytes are the offset within the segment and the second two bytes are the segment address.

An example of using the pointers from Basic:

```
10 DEF SEG=0
20 FONT=PEEK(&H0706)+256*PEEK(&H0707)
30 DEF SEG=PEEK(&0708)+256*PEEK(&H0709)
```

Statement 30 sets the variable FONT to point to the active character font, and statement 30 sets the current segment to the segment containing the font.

Address	Length (bytes)	Set by	Use
400H	1	Boot	Boot (P)ROM version number
401H	1	Boot	Machine type 0 = Apricot & Apricot XI 1 = Apricot Portable 2 = Apricot F1 & F1e
402H	2	Boot	RAM Memory Size (in paragraphs)
404H	4	Boot	Cold/Warm bootstrap mark (5678H,1234H)
408H	2	Boot	Drive booted from 0000H = Floppy 0 0001H = Floppy 1 0002H = Winchester 1 0003H = Winchester 2
40AH	4	Boot	Pointer to loaded boot disk header sector (immediately after Boot only)
40EH	4	Boot	Reserved
412H	2	Boot	Reserved
414H	1	Boot	Winchester type: 0 = No Winchester 3 = 5 Megabyte RO351 Winchester 4 = 10 Megabyte RO352 Winchester 5 = 20 Megabyte Winchester
415H	1	Boot	Floppy type: 0 = 70 track SS 1 = 80 track SS 2 = 80 track DS
416H	1	Boot	Number of Floppy Drives
417H	1	Boot	Number of Winchester Drives
418H	4	Boot	Pointer to Floppy BPB array Table
41CH	4	Boot	Pointer to Winchester BPB array Table
420H	2	Boot	Power-up Diagnostic Results Test failure number from boot diagnostic
422H	2	Boot	Paragraph address of start of user code area (for use by BIOS's laded from disk).
424H	4	Boot	ROM BIOS Stack Segment/Offset (for use by interrupt routines)

Address	Length (bytes)	Set by	Use
500H	8		Reserved
508H	8		Reserved
600H	5	Boot	Long jump to BIOS Control Device
610H	112	User	Reserved for MBASIC/GSX interface
700H	4	Boot	Pointer to internal BIOS config table
704H	2	Boot	Length of internal BIOS config table in bytes
706H	4	Boot/ User	Pointer to active character font
70AH	2	Boot/ User	Length of active font in bytes
70CH	4	Boot/ User	Pointer to master character font
710H	2	Boot/ User	Length of master font in bytes
712H	4	Boot/ User	Pointer to active keyboard tables
716H	2	Boot/ User	Length of active keyboard tables
718H	4	Boot	Pointer to internal keyboard tables
71CH	2	Boot	Length of internal keyboard tables
71EH	4	Boot/ User	MicroScreen Character Table pointer. Not used.
722H	4	Boot	Pointer to write only register copy table
726H	4	Boot	Pointer to Ascii colour screen image
72AH	4	Boot	Pointer to Ascii LCD screen image
780H	4	User	Voice software pointer area

Ascii and Bit Screen Image

Two areas of RAM are reserved for the Display. The Apricot features mixed graphics and text features and this is achieved by use of a Bit Screen Image.

Manipulation of the Bit Image for Scrolling, Hardcopy etc is inherently slow and cumbersome. Therefore a further image in the more suitable Ascii form is maintained.

A further area of the RAM contains hardware mapped video line pointers — one pointer per Pixel line, 256 in total. These pointers are used for scrolling etc.

The Bit Screen Image and video line pointers are fully described in the Screen Driver chapter.

RAM BIOS for MS-DOS

The function of the RAM BIOS is to link MS-DOS to the ROM BIOS. It does this by passing MS-DOS calls to the Control Device; i.e. interrupt FC.

Bootable Disks for MS-DOS 2.11 include a full generic RAM BIOS, i.e. it is designed to boot on all Apricots equipped with a ROM BIOS.

The RAM BIOS will support any machine configuration up to 2 Floppy Drives and 2 Winchester Drives.

The Fixed Position Files on the Bootable Disk are as follows:-

File	Start Sector (70T/ss)	Start Sector (80T/ds)	Length
FONT.SYS (16*16, 8*8 and 8*10)	0DH	12H	12.5K
KEYTAB.SYS	26H	2CH	1K
IO.SYS	28H	2EH	7K
MSDOS.SYS	36H	3CH	17K

Note that the size of MSDOS.SYS will increase for MS-DOS 3.00 to approximately 28K.

Disk Label Sector and Configuration Table

The Disk Label Sector for Generic Boot disks is situated in Track 0, Sector 0 on the boot disk (the first physical sector on the disk), and is 512 bytes long.

It contains both disk and operating system identification, boot loading information, and the ROM BIOS configuration table (this must be valid on the boot disk).

The first 80H bytes of the Label Sector contain disk and OS configuration data. The remainder, i.e. bytes 80H to FFH contains the configuration data for all other physical devices.

The Pointer Table has an entry at at 40AH which points to the loaded boot disk header sector image immediately after a BOOT is performed. This is for Operating system loader and initialise software only.

The data in the Label Sector is also stored partly in the Pointer Table and the remainder in the Configuration Table.

A further pointer at 700H points to the internal copy of configuration parameters (2nd 80H bytes of the disk header).

First 80H bytes, disk & O/S identification

Offset	Reference	Bytes	Description
0000	LBLform_vers	8	version of format which created disk
0008	LBLop_sys	1	Operating System: 0 = invalid 1 = MS-DOS 2 = p-System 3 = CP/M 86 4 = Concurrent CP/M
0009	LBLsw_prot	1	software write protect (0 = off)
000A	LBLcopy_prot	1	copy protect (0 = off)
000B	LBLboot_disk	1	boot disk type: 0 = non-bootable disk 1 = Apricot & XI RAM BIOS 2 = GENERIC ROM BIOS 3 = Apricot & XI ROM BIOS 4 = Apricot Portable ROM BIOS 5 = Apricot F1 ROM BIOS
000C	LBLmulti_region	1	multi-regioned 0 = not multi-regioned <>0 = number of logical volumes)
000D	LBLwinchester	1	Winchester disk (1 = winchester)
000E	LBLSec_size	2	sector size (in bytes)
0010	LBLsec_track	2	sectors per track
0012	LBLtracks_side	4	Tracks per side
0016	LBLsides	1	Sides: 1 = single 2 = double
0017	LBLinterleave	1	interleave factor
0018	LBLskew	2	skew factor
001A	LBLboot_locn	4	sector number of boot image
001E	LBLboot_size	2	number of bootstrap sectors

Offset	Reference	Bytes	Description
0020	LBLboot_addr	4	boot load address (dword pointer)
0024	LBLboot_st_off	2	boot start address offset
0026	LBLboot_st_seg	2	boot start address segment
0028	LBLdata_locn	4	sector number of first data block
002C	LBLgeneration	2	generation number
002E	LBLcopy_count	2	copy count
0030	LBLcopy_max	2	maximum number of copies
0032	LBLserial_id	8	serial number
003A	LBLpart_id	8	part number
0042	LBLcopyright	14	copyright notice
Main Disk Extended BPB image:			
0050	LBLBPBsctr_sz	2	sector size in bytes
0052	LBLBPBclu_sz	1	cluster size in sectors
0053	LBLBPBrsvd_sct	2	reserved sectors
0055	LBLBPBn_fats	1	number of FAT's
0056	LBLBPBn_dir_ent	2	number of directory entries
0058	LBLBPBn_sectors	2	number of sectors
005A	LBLBPBmedia_id	1	media ID byte
005B	LBLBPBn_fat_sct	2	number of sectors per FAT
005D	LBLBPBdisk_type	1	type of disk: 0 = 70 track SS 1 = 80 track SS 2 = 80 track DS 3 = 5M winchester 4 = 10M winchester 5 = 20M winchester
005E	LBLBPBstart_sct	2	Logical sector for start of volume
End of BPB image.			
0060	LBLfont_name	16	name of default FONT.SYS
0070	LBLkeys_name	16	name of default KEYTAB.SYS

Configuration constants definition area

(Base address = Label Sector base + 80H)

Note: Run-time ROM BIOS pointer at 700H points to BIOS copy of these configuration tables.

Offset	Reference	Bytes	Description
Systems unit:			
0080	CNF_ver_lo	1	Disk BIOS minor version number
0081	CNF_ver_hi	1	Disk BIOS major version number
0082	CNF_diagflag	1	Global diagnostics flag (0 = off)
0083	CNF_lst_dev	1	PRN: device 0 = parallel 1 = serial
0084	CNF_Bell_vol	1	bell volume 0 = full 15 = off)
0085	CNF_cache_on		Reserved
0086	CNF_graphics_on		Reserved
0087	CNF_DOS_len	1	Length of DOS in sectors
0088	CNF_FONT_len	1	Length of FONT in sectors
0089	CNF_KEYS_len	1	Length of KEY table in sectors
008A	CNF_DOS_sec	2	Start sector of DOS image
008C	CNF_FONT_sec	2	Start sector of 3 Fonts (12.5K)
008E	CNF_KEYS_sec	2	Start sector of KEY table image

Note: **The above six fields are private to the MS-DOS BIOS; other O/S's may use these for their own private uses. If FONT_sec or KEYS_sec are set to zero then the respective tables are not loaded, and BIOS uses the ROM versions.**

Offset	Reference	Bytes	Description
Keyboard:			
0090	CNF_Click_vol	1	Key click volume 0 = full 15 = off
0091	CNF_rept_en	1	Auto-repeat master enable 0 = off 1 = on
0092	CNF_rept_dly	1	Auto-repeat lead-in (not used)
0093	CNF_rept_int	1	Auto-repeat interval (not used)
0094	CNF_Mscrn_mode	1	Microscreen mode (not used)
0095		11	spare
Screen:			
00A0	CNF_line_mode	1	0 = 256 line 1 = 200 line display
00A1	CNF_line_width	1	0 = 80 1 = 40 column display
00A2	CNF_image_off	1	0 = Screen Image on 1 = image off
00A3	13		spare

Offset	Reference	Bytes	Description
Serial communications:			
00B0	CNF_Tx_brate	1	Tx baud rate 1 = 50, 2 = 75, 3 = 110 4 = 134.5, 5 = 150, 6 = 300, 7 = 600, 8 = 1200, 9 = 1800 10 = 2400, 11 = 3600, 12 = 4800, 13 = 7200, 14 = 9600, 15 = 19200
00B1	CNF_Rx_brate	1	Rx baud rate 1 = 50, 2 = 75, 3 = 110 4 = 134.5, 5 = 150, 6 = 300, 7 = 600, 8 = 1200, 9 = 1800 10 = 2400, 11 = 3600, 12 = 4800, 13 = 7200, 14 = 9600, 15 = 19200
00B2	CNF_Tx_bits	1	Tx bits per char. (5 to 8)
00B3	CNF_Rx_bits	1	Rx bits per char. (5 to 8)
00B4	CNF_stop_bits	1	stop bits 1 = 1 2 = 1.5 3 = 2
00B5	CNF_parity_chk	1	parity check 0 = no check 1 = check
00B6	CNF_parity_typ	1	parity type 0 = none 1 = odd 2 = even 3 = mark 4 = space
00B7	CNF_Tx_xonxoff	1	transmit xon/xoff protocol 0 = off 1 = on
00B8	CNF_Rx_xon/xoff	1	receive xon/xoff protocol 0 = off 1 = on
00B9	CNF_xon_char	1	XON character code
00BA	CNF_xoff_char	1	XOFF character code

Offset	Reference	Bytes	Description
00BB	CNF_Rx_X_limit	2	XON/XOFF receive buffer limit
00BD	CNF_dtr_dsr	1	DTR/DSR protocol 0 = off 1 = on
00BE	CNF_cts_rts	1	CTS/RTS protocol 0 = off 1 = on
00BF	CNF_CR_null	1	number of nulls to send after CR
00C0	CNF_FF_null	1	nulls (x10) to send after FF
00C1	CNF_s_cr_lf	1	Auto LF after CR 0 = off 1 = on
00C2	CNF_s_bioserr	1	BIOS error report 0 = off 1 = on
00C3		13	spare
Parallel communications:			
00D0	CNF_p_cr_lf	1	auto LF after CR 0 = off 1 = on
00D1	CNF_select	1	select line support 0 = off 1 = on
00D2	CNF_pe	1	paper empty support 0 = off 1 = on
00D3	CNF_fault	1	fault line support 0 = off 1 = on
00D4	CNF_p_bioserr	1	BIOS error report 0 = off 1 = on
00D5		11	spare

Offset	Reference	Bytes	Description
Keyboard:			
Winchester:			
00E0		14	spare
00EE	CNF_wini_park	1	parking enable flag 0 = on nz = off
00EF	CNF_wini_form	1	format protection 0 = off nz = on
RAM disk:			
00F0		16	spare
Non-dedicated area of label sector starting at base + 100H; used for Winchester disk bad block tables and Multi-volume BPB's.			
0100	WINbad_sect	64	Up to 32 words giving logical sector numbers of bad blocks on the disk
0140	WINvol_bpb1	16	Extended BPB volume 1
0150	WINvol_bpb2	16	volume 2
0160	WINvol_bpb3	16	volume 3
0170	WINvol_bpb4	16	volume 4
0180	WINvol_bpb5	16	volume 5
0190	WINvol_bpb6	16	volume 6
01A0	WINvol_bpb7	16	volume 7
01B0	WINvol_bpb8	16	volume 8
01C0		63	spare
01FF		1	CP/M sides flag (0 = single sided)

Control device 3.2

Contents

Overview

General application
- Introduction
- Low level Control device access
- High level Control device access
- Errors

Specific application
- Device Numbers
- Screen
- Keyboard
- Serial I/O
- Parallel I/O
- Mouse
- Clock
- Sound
- Floppy disk
- Winchester

Overview

The Control Device is a software driver which routes requests for I/O to the individual hardware drivers in the ROM BIOS. The philosophy behind this device is to provide a generic interface for the Apricot family.

Application software written using the Control Device will thus be compatible with future generations of the Apricot family.

Application software which detours the Control Device in order to access the hardware directly cannot assume compatability with future releases within the family. This practice is not to be condoned and should only be used in the last resort.

The Control Device is implemented fully on the F1 and Portable. At present only a subset is available on the pc and Xi but future versions will be brought into line with the inclusion of a full implementation of the generic Control Device.

Two methods of invoking the Control Device are provided to facilitate the needs of low and high level languages. These are described in detail in the next section.

An example of the low level requirement can be seen in the RAM BIOS which serves MS-DOS with all I/O functions. The RAM BIOS is implemented at low level machine code and it routes all I/O via the Control Device. High level languages such as BASIC may also make use of the Apricot features but do not have the same facilities to invoke the Control Device.

The following sections provide a blueprint for utilising the Control Device, a description of each of the hardware drivers together with examples of how to access them from a high level language.

Remember that the Control Device is a generic interface and that the inclusion of drivers in the following sections does not neccessarily imply that the hardware is available on a specific machine. The Microscreen is not available on the Portable for example but its driver is naturally a part of the generic family. Such occurences are noted and a description of the effect of calling a non-existant hardware module is included.

General application

Introduction

This section describes how to invoke the Control Device from low and high level languages.

The two methods each require the following arguments to be passed to and from the individual routines:

Entry:

P1 = Device number

P2 = Command

P3 = Data or Data pointer segment

P4 = Data or Data pointer offset

Exit:

PX = Data/Status

Reference to individual routines will determine the format of the parameters.

The first method is available at machine code level and is invoked by executing a software interrupt of type FC hex with the registers set up with the entry parameters.

The alternative method is available to facilitate languages which are not able to generate a software interrupt or indeed to access the processor registers.

This involves calling a fixed location in the Pointer area (0600 hex) which contains a "Long jump" to the Control Device interface routine with the parameters on the Stack. In Basic, for example, this is achieved with a CALL Statement.

Note that interpretive BASIC is used in this chapter and others purely as a universal quick reference to the use of facilities. It is not really a suitable language for accessing the Control Device and in certain cases still requires machine code subroutines to succeed. It is recognised that use of the Control Device will fall mainly into the category of the professional Application package developer using assembler, C, PASCAL, Compiled BASIC, etc.

Low level Control Device access

Entry:

Set the internal registers with the following "word" values:

BX = Device number (P1)

CX = Command (P2)

DX = Data or Data ptr segment (P3)

SI = Data or Data ptr offset (P4)

Call:

Execute software interrupt type FC hex

Exit:

AX = Data or Status (PX)

All routines in the Control Device preserve the entry registers. The AX register will not normally be preserved.

The above parameters and result are all "word" values. To facilitate functions which require additional data the DX register is combined with the SI register to provide a Segment:Offset pointer.

DX:SI = Segment:Offset

In this case the return data/status will normally be at the location pointed to by DX:SI.

The Device numbers are detailed in later sections. So too are the command values together with detail which will determine which routines require the "Data pointer" option and how each routine returns Data or Status.

High level Control Device access

Entry:

Set the following "word" pointers on the stack:

Device number (P1)

Command (P2)

Data (P3)

Result (P4)

Call:

Call the Control Device via the "Long Jump" in the Pointer area. Absolute address 00600 hex.

Exit:

Data or Status (P4) or at (P3:P4)

The interface for the Control Device expects the stack to contain 4 word pointers to each of the entry parameters given above. The order in which the parameters are to be pushed onto the stack is Device, Command, Data and finally Result.

Normally, as in BASIC for example, the language processor will be responsible for setting up the stack with the relevant pointers. The applications program must simply ensure that the parameters are supplied in correct order and number.

The DX:SI Data pointer facility is also available in High level access by combining the two pointers P3:P4 to give Segment:Offset respectively or Double word data.

It is of particular importance to note that pointers within most language processors, and once again BASIC is a good example, are relative to the processor's own Data Segment. There is usually no direct method for the Application to obtain the value of the Segment. In such cases the Application must resort to machine code subroutines to ensure a correct interface. Refer to the Appendix on Language Interfaces for details of how to do this and further information regarding interfaces.

Apart from these differences, the Control Device is accessed and reacts in the same way whether at Low or High level.

High level Control device access

Accessing the Control Device from BASIC

The following sections provide examples of how to use the Control Device from BASIC. Here we can study the general concept of implementing I/O functions from BASIC.

```
10 REM – Concept of programming Control Device
20 DEF SEG=&H60 'Set current segment to 60 hex
30 IO=0         'Pointer to offset 0 within segment
40 DEV%=1       'Device number 1
50 COM%=0       'Command is 0 i.e. initialise
60 DAT%=0       'Data
70 RET%=0       'Data/Status return
100 CALL IO(DEV%,COM%,DAT%,RET%)
```

The CALL will vector to absolute address 00600 hex.

The stack will contain the following:

Word pointers:	Device number	(DEV%) i.e.	P1
	Command	(COM%)	P2
	Data	(DAT%)	P3
	Return data	(RET%)	P4

"pushed" in the order given followed by a "Double word" return address.

The Control Device will validate the parameters and route the I/O request to the specified routine.

Execution of the command will normally result in Data/Status being returned to the application program via the RET% variable.

```
200 SEG%=&HC000 'Segment pointer to a buffer
210 OFF%=0      'Offset within buffer
220 DEV%=10:COM%=3
250 CALL IO(DEV%,COM%,SEG%,OFF%)
```

The example follows on to show the alternative entry parameters where the SEG% (P3) & OFF% (P4) variables represent a double word Segment:Offset pointer to the data.

Any return data or status is returned relative to the data pointer.

Note: The references to device numbers, commands etc, given in the examples are purely arbitrary. The exact specification for each individual driver is described below.

Errors

On exit from the Control Device, PX will be set to a value to indicate the status or data returned from the individual command. Where data is returned, details are given with each call.

Where a status is returned then it will be one of the following unless otherwise stated:

0000 - Device present and on-line
8000 - Write protect
8001 - Unknown unit
8002 - Not ready
8003 - Unknown command
8004 - CRC error
8005 - Bad request length
8006 - Seek error
8007 - Unknown media
8008 - Sector not found
8009 - Printer out of paper
800A - Write fault
800B - Read fault
800C - General failure
FFFF - Device error/invalid device or command

In certain cases the 0200 hex bit is set to indicate that the device is busy.

Specific application

This section details each individual hardware driver and how it is accessible via the Control Device.

The section is limited to the definition of each call. A full description of the driver and examples of how to use its functions from a high level language (and implicitly a low level language) are given in the following chapters on the Drivers.

The Device number is given with the heading of each driver and not included in the definition to avoid repetition. It must be stressed however that this parameter *MUST* be supplied with every call to the Control device.

For ease of identification and to provide a common approach to each method of invoking the Control Device, each parameter will be given an identifier as follows:

P1 - Device number

P2 - Command

P3 - Data or Data Pointer Segment

P4 - Data or Data Pointer Offset

PX - Data or Status

PTR - Pointer (Double word Segment:Offset)

In addition to the parameters above the following identifiers have a common meaning in the driver sections:

SEG - Segment
OFF - Offset
SET - Current setting.

A number of commands are of the format 'Get/Set an attribute' and successful execution results in the PX parameter being returned SET (i.e. the current setting). If P3 is invalid then the setting, which is returned in PX, remains unchanged.

Note: Invalid P3 settings may be conveniently used to obtain the current setting.

Certain calls in the Generic Control Device are not available on all machines within the family. Availability is denoted in the M/C (machine column):

A is available on Apricot or Xi
P is available on Portable
F is available on F1

Device numbers

The Device numbers are:

ASCII	Hex	Device
1	31	Screen
2	32	Keyboard
3	33	Microscreen (only on Apricot PC/Xi)
4	34	Serial I/O
5	35	Parallel I/O
6	36	Mouse
7	37	Clock
8	38	Sound
9	39	Floppy Disk
@	40	Winchester
A	41	Modem (not implemented in Control Device)
B	42	Cache/Graphics/IBM (only on Apricot PC/Xi)

Screen - P1 = 31 hex

M/C	P2	Function	P3	P4	PX	Comment
PFA	0000	Initialise driver			0000 OK 0001 No driver	
A	0001	Get/Set Text mode Get/Set Graphics mode	0000 0001		SET SET	note 1
A	0002	Get/Set display on Get/Set display off	0000 0001		SET SET	note 2
	0003	No-op				
PF	0004	Print character 'cc'	00cc			note 3
PF	0005	Print string	SEG	OFF		note 4
PF	0006	Reserved				
PF	0007	Reserved				
PF	0008	Reserved				
PF	0009	Reserved				
PF	000A	Reserved				
PF	000B	Reserved				
PF	000C	Reserved				
PF	000D	Reserved				
PF	000E	Output char. to Default screen.	rrcc	FORMX		note 5
PF	000F	Update Screen Bit Image from Default character Image				note 5/6

Key: rrcc rr = screen row
cc = screen column

Screen driver notes

1. Only available on pc/Xi where text mode is a feature.
2. Switch Display OFF/ON. Suitable for pseudo rapid Display changes.
3. The use of software Interrupt F1 hex is faster. This is a logical print, i.e. characters < 20 hex are obeyed.
4. SEG:OFF is a pointer to a 3 word table:

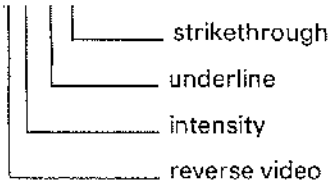
   Word 0 - Number of characters
   Word 1 - Offset of string
   Word 2 - Segment of string

5. FORMX - A word to represent the current Default Screen. Three formats are possible:

   Apricot Compatible 80 column:

   Lo byte - 8 bit ASCII data

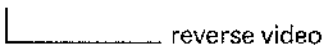
   Hi byte - r i u s x x x x

   Apricot 40 column:

   Lo byte : 8 bit ASCII data

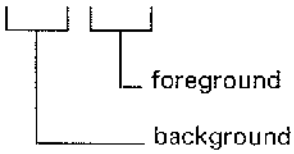
   Hi byte : r x x x x x x x

   Colour:

   Lo byte : 8 bit ASCII data

   Hi byte x b b b x f f f

   Refer to Screen Driver chapter for a definition of Apricot Compatible mode.

6. This is a rapid update of the Screen Bit Image from the Default Screen Character Image. It is used by the Screen Driver following such functions as scrolling.

Keyboard - P1 = 32 hex

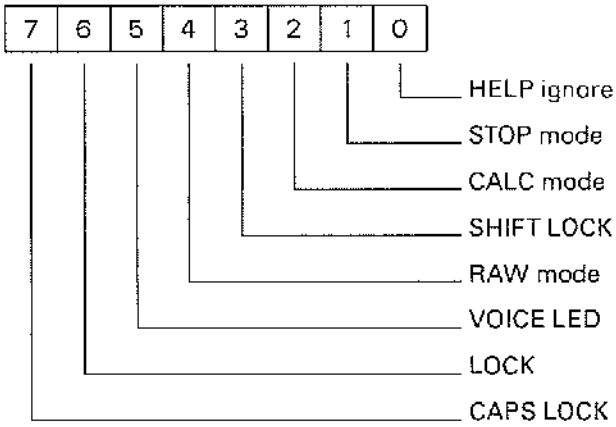
M/C	P2	Function	P3	P4	PX	Comment
PFA	0000	Initialise			0000 0001	OK no driver
PFA	0001	Disable auto rpt Enable auto rpt	0000 0001		SET SET	
A	0002	Set auto rpt lead-in delay to xx 20ms intervals	00xx		SET	
A	0003	Set auto rpt rate to xx 20 ms	00xx		SET	
PFA	0004	Set Fall-through mode : OFF ON	0000 0001		SET SET	note 1 note 2
PFA	0005	Flush queue			0000	note 3
PFA	0006	Reset CTRL/SHIFT status & clear down-code buffer			0000	note 4
PFA	0007	Get/Set HELP ignore mode: OFF ON	0000 0001		SET SET	
PFA	0008	Place data xx in queue		00xx	0000 FFFF	OK Full/BELL
PFA	0009	Sound BELL			0000	
PFA	000A	Return Queue byte count			00xx	xx = number
PF	000B	Get byte from Queue Wait if queue empty			00xx	xx = byte
PF	000C	Look-ahead. Get Next byte.			00xx FFFF	xx = byte empty
PF	000D	Add string to queue	SEG	OFF	0000 FFFF	OK - note 5 Full/BELL
PF	000E	Get string from queue	SEG	OFF	0000	note 6
PF	000F	Get/Set Status	aaoo		00xx	note 7

Keyboard notes

1. Down-codes are translated and then put in the queue.
2. Down-codes are not translated. They are placed in the queue "raw".
3. Remove all data from queue.
4. Set CTRL and SHIFT to not affective and clear down codes in queue. Use after Keyboard Table change.
5. SEG:OFF is a pointer to a 3 word packet consisting:

   WORD = No of bytes in string (max 80 - no check)
   DWORD = Seg:Off pointer to string

   If there is not enough room for the complete string in the buffer then it is rejected, the bell is sounded and the status is set to FFFF hex.
6. As 5 except for remarks on overflow. The byte count in this case must be pre-specified and the call will not return until the specified Key count.
7. The status of the keyboard is configured in a byte as follows:



where 0 = off 1 = on

The high byte of P3 (aa) is logically AND'ed with the status, i.e. gets the current status, whereas the low byte (oo) is logically OR'ed with the status and hence sets it.

e.g P3 = FF00H will return status only
P3 = FD02H will set STOP on

Serial I/O - P1 = 34 hex

M/C	P2	Function	P3	P4	PX	Comment
PFA	0000	Initialise driver			0000	
PFA	0001	Xmit character xx	00xx		0000 FFFF	OK Fail
PFA	0002	Rcve character			FFFF 00xx	char avail. OK
PFA	0003	Update SIO settings			0000	note 1
PFA	0004	Get/Set Xmit baud rate (x = 1 - F)	000x		SET	note 2
PFA	0005	Get/Set Rcve baud rate (x = 1 - F)	000x		SET	note 2
PFA	0006	Get/Set Xmit bits/char. (x = 5 - 8)	000x		SET	note 3
PFA	0007	Get/Set Rcve bits/char. (x = 5 - 8)	000x		SET	note 3
PFA	0008	Get/Set Stop bits/char. (x = 1 - 3)	000x		SET	note 4
PFA	0009	Get/Set Parity type. (x = 0 - 2)	000x		SET	note 5
PFA	000A	Get/Set Xmit XON/XOFF	000x		SET	0 = disable 1 = enable
PFA	000B	Get/Set Rcve XON/XOFF	000x		SET	0 = disable 1 = enable
PFA	000C	Get/Set RTS (Ready to send)	000x		SET	0 = reset 1 = set
PFA	000D	Get/Set DTR (Data Term ready)	000x		SET	0 = reset 1 = set
PFA	000E	Get/Set Xmit enable	000x		SET	0 = disable 1 = enable
PFA	000F	Get/Set Rcve enable	000x		SET	0 = disable 1 = enable note 6
PFA	0010	Get CTS (Clear to send) status			0000 0001	Reset Set
PFA	0011	Get DCD (Data Carr. Detect)			0000 0001	Reset Set
PFA	0012	Get DSR (Data Set Ready) status			0000 0001	Reset Set
PFA	0013	Get/Set xx nulls after CR	00xx		SET	note 7
PFA	0014	Get/Set xx nulls after FF	00xx		SET	note 7
PFA	0015	Get/Set auto LF after CR	000x		SET	0 = disable 1 = enable

Serial I/O notes

1. All calls to configure the SIO are not activated until this command is executed. Specifically these calls are 0004 through 0009, 0018 and 0019.

2. P3 is set to a value (x) in the range 1 - F hex corresponding to an entry in the table below:

Value	Baud Rate
1	50
2	75
3	110
4	134.5
5	150
6	300
7	600
8	1200
9	1800
A	2400
B	3600
C	4800
D	7200
E	9600
F	19200

3. P3 is set to a value (x) in the range 5 - 8 to denote the number of data bits per character.

4. P3 is set to a value (x) in the range 1 - 3 corresponding to an entry in the table below:

1 - 1 stop bit
2 - 1.5 stop bits
3 - 2 stop bits

5. P3 is set to a value (x) in the range 0 - 2 corresponding to a parity type as given in the table below:

0 - none
1 - odd
2 - even

6. These calls directly enable/disable the SIO. Care must be taken to ensure that the SIO has completed all outstanding communications. This can only be acieved by either monitoring the Read Registers directly and looking at the "ALL SENT" flag or by use of Call 24 hex which does not return 0 until "ALL SENT".

7. P3 is set to a value (xx) in the range 0 to FF hex. This command is of use in conjunction with transmission to printer devices. It generates a series of null characters (10 times the xx number specified) following a CR or FF as designated.

Serial I/O - P1 = 34 hex (continued)

M/C	P2	Function	P3	P4	PX	Comment
PFA	0016	Enable serial mouse			0000	note 8
PFA	0017	Disable serial Mouse			0000	note 8
PFA	0018	Get/Set RTS/CTS protocol	000x		SET	0 = disable 1 = enable 2 = auto-enable note 9
PFA	0019	Get/Set DTR/DSR protocol	000x		SET	note 10
PA	001A	Get/Set External SIO control	000x		SET	note 11
PF	001B	Rcve Queue Look-ahead			FFFF 00xx	none Rcve char.
PF	001C	Flush Xmit Queue			0000	note 12
PF	001D	Flush Rcve Queue			0000	note 12
PF	001E	Get Rcve Char. Status-reset error.F			00xx	note 13
PF	001F	Xmit string	SEG	OFF	0000 xxxx	OK Error code Note 14
PF	0020	Rcve string	SEG	OFF	0000	Note 14
PF	0021	Get/Set Rcve Queue length	xxxx		SET	Note 15
PF	0022	Get/Set Xmit Queue length	xxxx		SET	Note 15
PF	0023	Return count chars in RCVE Queue			xxxx	
PF	0024	Return count chars in XMIT Queue			xxxx	

Serial I/O notes (continued)

8. When the serial mouse is enabled the SIO interrupt handler vectors all RX data to the Serial Mouse Interrupt (0FA hex) handler. Control is returned to the RS-232 when the serial mouse is disabled.

9. RTS/CTS and DTR/DSR protocols are supported by the SIO directly when Auto-enables mode is invoked. In Auto-enables mode the CTS line serves a specific function, i.e. no transmission takes place until CTS goes low. If CTS is used for any other purpose then Auto-enable should not be invoked.

10. Refer to note 9. In addition both enable/disable in this call de-activates Auto-enable in both protocols.

11. This command allows the application to switch the SIO between BIOS control and user control. The SIO under BIOS, for example, is only in ASYNC - in order to program the SIO in SYNC mode the user must take control of the SIO and re-program it. The entry parameter (P3) selects one of the following:

    0 - SIO under BIOS control
    1 - SIO under interrupt control (F0 hex)

    In the latter case the interrupt routine F0 hex must handle all Tx, Rx and errors related to the SIO.

    The return cell (PX) gives details of the SIO:

    PX low byte - Base SIO port address

    The offsets from the base are:

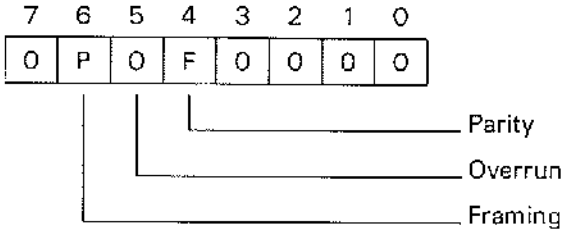
    [base + 0] = Ch A data
    [base + 2] = Ch A status
    [base + 4] = Ch B data
    [base + 6] = Ch B status

    PX high byte - has one of 2 values i.e.

    0 = RS-232 on Ch A
    1 = RS-232 on Ch B

12. These calls should be invoked if spurious data is transmited/received.

13. Returns status in PX low byte as per SIO Read Register 1 (refer to Hardware section).
    This is a 'running' error status on all RX characters since the last status call. The call clears the status.



    The status is used to detect errors in RX Blocks of characters.

    Note that all other bits are clear. The call cannot be used to monitor 'ALL SENT'.

14. Parameters P3 & P4 form an address pointer SEG:OFF to a 3 word packet of the format:

    WORD - number of bytes in string
    DWORD - pointer to string (SEG:OFF)

15. Set the Queue lengths to a value xxxx in the range 1 - 512 bytes.

Parallel I/O - P1 = 35 hex

M/C	P2	Function	P3	P4	PX	Comment
PFA	0000	Initialise			0000	
PFA	0001	Return available buffer space.			xxxx	note 1
PA	0002	Get/Set FAULT line status detect.	000x		SET	0 = disable 1 = enable note 2
PA	0003	Get/Set SELECT line status detect.	000x		SET	0 = disable 1 = enable
PA	0004	Get/Set PAPER OUT status detect.	000x		SET	0 = disable 1 = enable
PFA	0005	Get/Set Auto LF after CR enable	000x		SET	0 = disable 1 = enable
PFA	0006	Get/Set Serial/Parallel Device	000x		SET	0 = Parallel 1 = Serial note 3
PFA	0007	Print character on output device.	00cc		xxxx	cc = character PX = status note 4
PF	0008	Flush printer output buffer			0000	note 5
PF	0009	Transmit string	SEG	OFF	xxxx	PX = status note 6
PF	000A	Return output status			0000 xxxx	OK Printer busy Paper out

Parallel I/O notes

1. The buffer has a capacity of 2k bytes. If the value xxxx in PX is less than 3 then regard as full.

2. Detection of the 3 error lines:

   FAULT
   SELECT
   PAPER ERROR

   may be individually enabled or disabled.

3. By default the BIOS is vectored to the parallel port. This toggle switch allows re-direction to the Serial port. In doing so the parallel printer buffer (2 Kbytes) is then used for Serial Output.

   Serial output is mixed with parallel output.

4. If the character is received by the current output support driver and successfully output the PX is returned zero. Otherwise PX contains an error code.

5. Resets pointers to beginning of buffer. Ensure that printing has been completed under normal circumstances by querying the number of free bytes in the buffer.

6. P3:P4 form a pointer to a 3 word packet as follows:

   WORD - byte count
   DWORD - string pointer

7. If the printer is busy or Paper out detected then PX will be set to an error code.

Mouse - P1 = 36 hex

The Mouse is not directly supported by the Control Device. It is a loadable Device driver.

However calls to the Control Device are provided to enable and disable the Serial Mouse and these can be found in the section Serial I/O.

Clock - P1 = 37 hex

M/C	P2	Function	P3	P4	PX	Comment
PF	0000	Initialise-reset time to 0:00:00 1-1-80			0000	
PF	0001	Set Date/Time	SEG	OFF	0000	
		P3:P4 points to data block: WORD - Days since 1-1-80 BYTE - Minutes BYTE - Hours BYTE - Hundredths of secs. BYTE - Seconds				
PF	0002	Get Date/Time	SEG	OFF	0000	
		P3:P4 points to data block as command 2.				

Sound - P1=38 hex

M/C	P2	Function	P3	P4	PX	Comment
PFA	0000	Initialise			0000	
PFA	0001	Get/Set Key click volume	000x		SET	note 1
PFA	0002	Get/Set Bell volume	000x		SET	note 1
PA	0003	Get/Set Channel 2 volume	000x		SET	note 1
PA	0004	Get/Set Channel 3 volume	000x		SET	note 1
PA	0005	Get/Set Channel 2 frequency	0xxx		SET	note 2
PA	0006	Get/Set Channel 3 frequency	0xxx		SET	note 2
PFA	0007	Get/Set Bell frequency	0xxx		SET	note 2
PFA	0008	Get/Set Bell duration	00xx		SET	note 3
PF	0009	Sound Bell			0000	
PF	000A	Sound Click			0000	

Sound notes

1. The value of x determines the volume within a scale of 0 to 15 where 0 = maximum and 15 = minimum (figures in decimal).
2. The value of xxx determines the frequency within a scale of 0 to 1023 decimal where 0 = maximum and 1023 is minimum.
3. The value of xx determines the duration of the Bell sound in multiples of 20 ms. The value of xx is in the range 0 to 255 decimal.

Floppy disk - P1 = 39 hex

M/C	P2	Function	P3	P4	PX	Comment
PFA	0000	Initialise driver			xxxx	note 1
PFA	0001	Set Drive number	000x		FFFF SET	Invalid drive 0 = drive 0 1 = drive 1
PFA	0002	Set segment address of Track Image for format	SEG		0000	note 2
PFA	0003	Set offset address of Track Image for format	OFF		0000	note 2
PFA	0004	Format floppy disk x (0 = 70ss/1 = 80ss/2 = 80ds)	000x		0000 xxxx	OK Error status note 1
PFA	0005	Return status of disk x	000x		xxxx	note 3
PFA	0006	Set disk x status to swapped	000x		0000 FFFF	OK invalid drive
PFA	0007	Return disk type in drive x	000x		0000 0001 0002	70 track SS 80 track SS 80 track DS notes 1/5
PF	0008	Return drive type of drive x	000x		0000 0001 0002	70 track SS 80 track SS 80 track DS note 1/5
PF	0009	Check if disk swapped in drive x	000x		0000 0001 0002	not swapped swapped no disk note 4
PF	000A	Get BPB (do this whenever disk is swapped)	SEG	OFF	xxxx	PX = Status note 6
PF	000B	Read/Write/Verify/ Write + Verify	SEG	OFF	xxxx	PX = Status note 7
PF	000C	Return status of drive x	000x		xxxx FFFF	Status Invalid note 8

Floppy disk notes

1. Status - Refer to the Errors section above for a list of errors and status settings.
2. Track image is controller dependant. Refer to Hardware section.
3. Apricot Compatible version will - not be supported in future releases. Returns the status register of the device. Refer to the hardware section.
4. This command clears the "swapped" flag! *IMPORTANT - refer to the Disk Driver chapter Applications Interest section.*
5. If the unit is not identifiable then the status returned is 8001 hex.
6. Whenever a disk has been swapped then this call should be executed. The P3:P4 parameters sepcify a Segment:Offset pointer to a 3 word packet:

   WORD - Drive number (0 or 1)
   DWORD - Pointer to a 512 byte scratch area

   The status PX will be zero if the command was executed correctly otherwise refer to the section on Errors above for detail. *IMPORTANT - refer to the Disk Driver chapter Applications Interest section.*
7. The P3:P4 parameters form a pointer to a block of 6 words which has the following format:

   WORD - Drive number (0 or 1)
   WORD - Command
   0 = READ
   1 = WRITE
   2 = VERIFY
   3 = WRITE with VERIFY
   WORD - Address of first logical sector
   WORD - Number of sectors to transfer
   DWORD - Buffer address (not needed for Verify)

   The number of sectors field is set to the number of sectors actually transferred.

   The status PX will be zero for a successful operation otherwise refer to section 2.4 Errors for detail.
8. THIS CALL SUCCEEDS CALL 0005. It should be used in preference to it! Status is as for CALL 0005 - refer to Hardware section.

Winchester - P1 = 40 hex

M/C	P2	Function	P3	P4	PX	Comment
PF	0000	Initialise drivers			xxxx	Note 1
PF	0001	Set Disk x status to swapped	000x		0000 FFFF	OK Invalid drive note 2
PF	0002	Return drive type of drive x	000x		0000 0001 0002	5 megabyte 10 megabyte 20 megabyte note 3
PF	0003	Check if drive x disk swapped	000x		0000 0001 0002	Not swapped Disc swapped No disk note 2
PF	0004	Get BPB (do this whenever disk is swapped)	SEG	OFF	xxxx	PX = Status note 4
PF	0005	Read/Write/Verify/Write+Verify	SEG	OFF	xxxx	PX = Status note 5

Winchester notes

1. Status - Refer to the section on *Errors* for a list of errors and status settings.
2. The "swapped" status on Winchester is used to indicate a change in state, e.g. the Winchester has been formatted.
3. If the unit is not identifiable then the status returned is 8001 hex.
4. Whenever a disk has been swapped then this call should be executed. The P3:P4 parameters sepcify a Segment:Offset pointer to a 3 word packet:

   WORD - Drive number (0 or 1)
   DWORD - Pointer to a 512 byte scratch area

   The status PX will be zero if the command was executed correctly otherwise refer to section on Errors for details. IMPORTANT - refer to the Disk Driver chapter section Applications Interest.

5. The P3:P4 parameters form a pointer to an a block of 6 words which has the following format:

   WORD - Drive number (0 or 1)
   WORD - Command
   0 = READ
   1 = WRITE
   2 = VERIFY
   3 = WRITE with VERIFY
   WORD - Address of first logical sector
   WORD - Number of sectors to transfer
   DWORD - Buffer address (not needed for Verify)

   The number of sectors field is set to the number of sectors actually transferred.

   The status PX will be zero for a successful operation otherwise refer to section on Errors for detail.

Screen driver 3.3

Contents

Overview

Application interest
- Screen images
- Using ESCape sequences
- Screen environment
- Apricot compatible mode
- Colour
- ANSI ESCape sequences
- Windows and Cursor addressing
- Fonts
- Ascii control codes
- ESCape Sequence Table

Systems interest
- Screen Bit Image
- Character attributes
- 40 Column mode
- Scrolling
- Configuration table

Overview

The Apricot F1 supports either monochrome or colour Displays.

The Screen driver provides the following Display environments:

1. Apricot Compatibility in monochrome, or
2. 4 Colour Display

The hardware provides for 16 colours but this however limits the screen size to a maximum of 40 columns. The Screen driver limits the choice of colours to 4 from 16 to provide an 80 column by 25 row environment.

The Hardware section of this manual provides a complete description of the Screen RAM. Applications wishing to implement features currently outside the support of the BIOS, such as 16 colour, should reference this section.

The Screen driver provides a Generic interface for applications through the Control Device and the use of ESCape sequences.

Apricot compatibility is provided in monochrome only. The character attributes supported are Reverse, Strikethrough, Underline and Intensity.

The Screen environment, apart from the features mentioned above, also provides 2 further features to cater for the market. They are:

1. 40 or 80 column mode
2. 200 or 256 line mode

The 40 column mode is designed to support TV screens in the environments detailed above. Alternatively, the Application itself can provide the facilities for full 16 colour 40 column output, if required.

The 256/200 line mode provides for 50/60 Hz electrical standards. The Screen driver supports this feature by using different size fonts.

Other features available in the Screen Driver provide the Application with complete Display control. The principal ones are tabled below:

A comprehensive set of ESCape sequences including a subset of the ANSI standards.

A 16 Colour programmable palette, with the Driver provides support for the following:

1. Monochrome (2 colours from 16)
2. Colour (4 colours from 16)

Foreground, Background character and Screen base Colour selection.

Partial Windows.

Hard copy.

WP primitives e.g. L & R scrolling, Multi character insertion, etc.

Switchable character Fonts. Two Fonts are provided as standard to cater for the line modes outlined above. Applications may use their own character Fonts by loading them and designating them active.

Finally, the Apricot F1 features mixed text and graphics by utilising a Bit oriented Screen RAM. For this reason the Screen driver uses a separate character Image in the memory as described in the section below.

The following sections detail the above features and provide examples to assist in explanation.

Applications interest

Screen images

The Display has an area reserved in memory called an "Image" screen. The location of the Image can be found from an entry in the Pointer table.

The image consists of a "word" for every location on the screen. Each one contains the Ascii value of the character together with its attributes.

The attributes available vary according to whether the Screen Driver is in Apricot compatible monochrome mode or in colour. Later sections detail these attributes.

The image is used mainly to support hardcopy screen dumps and "return of the character at the current cursor position".

The real screen RAM is in another location and contains a bit image for each character. In order to distinguish between the two images the real screen RAM is termed the "Bit Image" whereas the character image is termed the "Screen Image".

The Bit Image is generated using the active Font table. This is discussed further under Systems interest.

When requests are made to output data to the screen then both the Screen Image and the Bit Image are updated by the Screen driver.

The Control Device provides calls which enable the Screen driver and Applications to compose a screen together with its attributes and then in a single command update the Bit Image from the Screen Image.

Using ESCape sequences

The ESCape sequences are detailed in two sections within the manual. Appendix E lists them in quick reference form, broken down by activity. They are also detailed in a later section as a Table in ascending Ascii order.

The ESCape sequences provide the main control features on the screen such as:

1. Screen characteristics e.g. intensity, underline ...
2. Cursor control e.g. position, UP, DOWN, LEFT ...
3. Colour e.g. set palette, fore/background ...
4. WP primitives e.g. Insert line, Delete line ...
5. Hard copy
6. ANSI, a subset of the standard sequences
7. Fast and slow screen scrolling

The ESCape sequences form a Generic interface across the Apricot family. For Applications writers wishing to produce compatible software, each ESCape is annotated with one or more of the following keys which indicates whether the ESCape sequence is supported on the particular machine:

P - Portable
F - F1
A - Apricot PC & Xi

Examples of how to use ESCape sequences are to be found throughout the manual. For readers unfamiliar with the principles, a full explanation can be found at the beginning of the section on ESCape sequences. To assist in the reading of the following sections a short example is provided overleaf:

Example: How to use an ESCape sequence

ESCape sequences consist of a series of predefined characters which are sent to the Screen driver by the Application. In BASIC this is done using the conventional PRINT statement. The sequence is prefixed with a special character, the ESCape character, whose decimal value is 27. It is not a printable character.

The next character in the sequence is the command. This in turn may be followed by one or more characters depending upon the individual command.

Usually these sequences make programs difficult to read and tend to clutter examples. For this reason examples in this manual define the command sequence in a string which is then given a meaningful name.

Each string may then be used freely with or without parameters as individual commands dictate.

```
10 CLS$=CHR$(27)+"E"      'Clear screen command (no parameters)
20 ENV$=CHR$(27)+"7"      'Set environment (needs 1 parameter)

100 PRINT CLS$            'Clear the Screen

110 PRINT ENV$+"2";"40 column Display"
```

Statements 10 and 20 pre-define the commands for the reason stated above. Statement 100 executes the Clear screen ESCape command and then statement 110 invokes the 40 column environment. Note that this requires an additional parameter of "2".

The full meaning of the Screen environment is discussed in the next section and uses the definitions formed above.

For readers not familiar with using ESCapes the long hand way of writing statement 110 is given below.

```
110 PRINT CHR$(27)+"7"+"2";"40 column Display"
```

In the short term there would seem to be little advantage in using the methods given above. However, more complex examples later in the manual demonstrate the benefits in terms of readability.

Screen environment

The screen environment may be modified at run time by invoking a selection of ESCape sequences. These enable Applications to select colour, width of the Screen and Apricot compatibility.

There are 4 modes which may be selected to provide the following environments:

ENV$+"0" 80 column Apricot monochrome compatability.

The Cursor is Homed and any image already on the Screen remains unchanged.

All Apricot attributes are supported together with background and foreground colours.

ENV$+"1" 80 column 4 Colour

Apricot Compatibility is not supported.

ENV$+"2" 40 column 4 Colour

This mode is designed for use with TV's. Only one Apricot attribute is supported - Reverse.

ENV$+"3" 80 column Apricot monochrome compatability. As ENV$+"0".

Reference to the Keyboard driver chapter shows that these sequences have been implemented in the default version of the keyboard table i.e. the environment may be set by pressing certain keys.

The examples in BASIC later in the section illustrate the use of these ESCape sequences.

The display may be driven in either 200 or 256 scan line mode, which is determined at BOOT time by a Configuration constant and cannot be modified during run time.

These line modes are supported by the incorporation of two character Font tables within the user RAM.

The Font options are:

1. 8 x 8 for the 200 line
2. 8 x 10 for the 256 line

The BOOT routines select the Font according to the constant in the configuration table and modify the pointers to reflect the "active" Font. The screen is driven by default in 80 column Apricot Compatible mode.

Apricot compatible mode

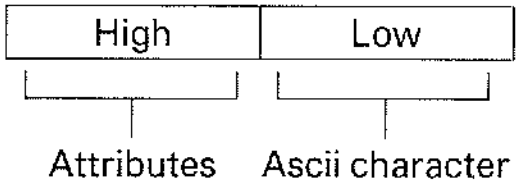
The Apricot compatible mode supports the following features:

1. Monochrome
2. Character attributes:

   Reverse video
   Intensity
   Underline
   Strikethrough
3. Word Processing primitives.
4. Screen dump
5. Partial windows

Apricot compatible mode is not available when the Screen environment is set to full colour, i.e. 4 from 16 colours.

The Screen Image has one word for each character on the screen which holds the Ascii value of the character together with it's attributes as illustrated below:

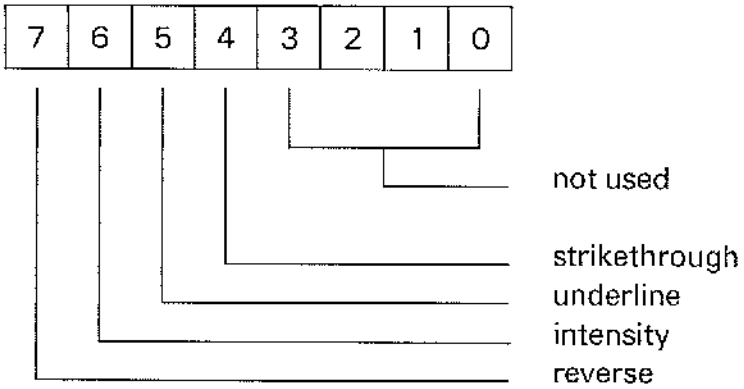


The attributes are used to define either the character feature stated above or the definition of background/foreground colour but not both together.

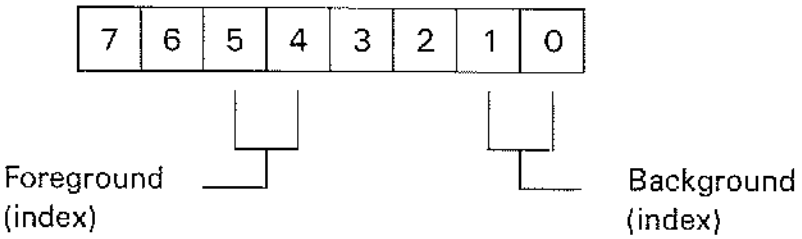
In 80 column Apricot compatible mode, any two colours are supported in monochrome together with standard character attributes. The attributes are not used to define the colours in this case but are reserved for normal text attributes.

The attributes have the following settings:

APRICOT compatible -



Colour -



The index is a reference to the colour palette. This is discussed in the next section.

Colour

The colour palette provides a selection of 16 colours. Each colour is designated a Colour code which is an Ascii value as given below:

Colour	Ascii Colour code
Black	0
Blue	1
Green	2
Cyan	3
Red	4
Magenta	5
Brown	6
Light Grey	7
Dark Grey	8
Light blue	9
Light green	:
Light cyan	;
Light red	<
Light magenta	=
Yellow	>
White	?

Each entry in the palette is referenced by an index which is also an ASCII value. The palette is assigned default values on Screen driver initialisation. The default settings for each index are as follows:

Ascii index	Ascii Colour code	Colour
0	9	Light Blue
1	?	White
2	0	Black
3	<	Light Red
4	:	Light Green
5	>	Yellow
6	;	Light Cyan
7	=	Light Magenta
8	1	Blue
9	7	Light Grey
:	8	Grey
;	4	Red
<	2	Green
=	6	Brown
>	3	Cyan
?	5	Magenta

The hardware configuration and screen environment dictate how many colours may be used, i.e.

2 for Apricot Compatible monochrome

4 for Colour

The significant colours in each of these are taken to be in the first x entries of the palette where x = 2 or 4.

Note:

1. In all environments the first entry (index "0") is the basic background colour referred to later as the "screen colour".
2. In the 2 colour Apricot compatible mode the following two entries have a significance:

   index "0" - Background
   index "1" - Foreground

3. The above default settings provide an acceptable contrast for all Applications regardless of the Display attached.

   This includes the ACT monochrome monitor which provides a scale of 8 shades of grey. In this case the I bit of the IRGB outputs is not used.

The setting of the palette is achieved via ESCape sequences, as the following example shows:

Example: Apricot compatible and full colour

```
10 REM - setting the Colour palette
20 CLS$=CHR$(27)+"z"          'clear screen
30 ENV$=CHR$(27)+"7"          'environment ESC
40 RES$=CHR$(27)+"xG"         'reset palette
50 COL$=CHR$(27)+"]"          'set palette ESC
70 FORE$=CHR$(27)+"5"         'foreground ESC
80 BACK$=CHR$(27)+"6"         'background

100 PRINT CLS$;RES$;ENV$;"3"  'Apricot compatible 2 colour.

110 PRINT COL$;"01";          'index "0" blue
120 PRINT COL$;"1>";          'index "1" yellow
130 PRINT "Yellow on Blue background"

140 INPUT "Press return key"; A   'just to pause

200 PRINT CLS$;RES$;ENV$;"1"  '4 colour ' Port.- 8 colour

300 PRINT COL$;"02";          'screen base green
310 PRINT COL$;"14";          'index 1 red
320 PRINT COL$;"29";          'index 2 light blue
330 PRINT COL$;"3>";          'index 3 yellow

400 PRINT BACK$;"1";          'char. background red
410 PRINT FORE$;"3";          'char. foreground yellow

500 PRINT "Yellow characters on Red background"
510 INPUT "Press return"; A   ' just a pause

600 PRINT BACK$;"3"           'char. background yellow
610 PRINT FORE$;"2"           'char. fore. light blue

700 PRINT "Light Blue chars. on yellow background"
```

Note:

1. Statement 100 sets the environment to 2 colour Apricot compatible mode. The background and foreground colours are defined in statements 110 and 120 respectively.
2. Statement 200 switches the environment to 4 colour.

To illustrate the above programming example further the palette is shown below in its reprogrammed form for the two different settings generated in the example.

Palette after statement 110 & 120:

Ascii index	Ascii Col. code	Colour
0	1	Blue
1	>	Yellow
2 thru ?	0	unchanged
	5	unchanged

Palette after statement 330 is:

Ascii index	Ascii Col. code	Colour
0	2	Green
1	4	Red
2	9	Light Blue
3	>	Yellow
4 thru ?	:	unchanged
	5	unchanged

Note: Index 0 is the Screen base colour.

ANSI ESCape sequences

A subset of the ANSI escape sequences is supported by the generic drivers through the use of ESCape sequences. The format of an ANSI ESCape sequence is:

```
CHR$(27)+"["+ parameters
```

The functions available are:

Mnemonic	Param.	Code	Function
CU			Cursor movement
CUU	n	A	n lines UP
CUD	n	B	n lines DOWN
CUF	n	C	n columns forward
CUB	n	D	n columns back
CUP	y;x	H	position cursor at y;x
HVP	y;x	f	position cursor at y;x (IBM compatible)
ER			Text erase:
ED	0	J	Erase-cursor to end of screen
ED	1	J	Erase-start screen to cursor
ED	2	J	Clear screen. Cursor not moved.
EL	0	K	Erase-cursor to end of line
EL	1	K	Erase-cursor to start of line
EL	2	K	Erase-entire line of cursor
CP			Report functions:
CPR	1;c	R	Cursor position to keyboard buffer
SM			Set mode:
LNM	20	h	Auto CR (on receipt of LF)
DECOM	6	h	Origin mode on
DECAWM	7	h	Auto-wrap at end of line
RM			Reset mode:
LNM	20	l	No auto CR
DECOM	6	l	Origin mode off
DECAWM	7	l	No auto-wrap
TAC			Text attributes
SGR	0	m	All attributes off
SGR	1	m	Bold ON
SGR	4	m	Underline ON
SGR	7	m	Reverse ON
SCR			Screen size/mode

Mnemonic	Param.	Code	Function
DSR	6	n	Device status report
DECSTBM	T;B	r	Set T (top) and B (bottom) lines
SCP		s	Save cursor position
RCP		u	Restore cursor position

The ANSI ESCape sequence parameters are required in the order:

CHR$(27) + "[" + 'param list' + 'Code'

Where there is more than one parameter a list is required with each parameter separated by a semi-colon.

Example: Cursor movement

```
10 AN$=CHR$(27)+"["        'ANSI lead-in
20 PRINT AN$;"2J"          'clear the screen
30 PRINT AN$;"10;15H";
40 PRINT "print from line 10 column 15"
50 PRINT AN$;"5A";
60 PRINT "now from line 5 column 1"
70 PRINT AN$;"10;14r";
80 PRINT "screen 5 lines high i.e lines 10-14"
```

Windows and Cursor addressing

A window may be designated in any rectangular block on the screen. The remainder of the screen remains unchanged.

All Print, scrolling and WP primitives will restrict themselves to the bounds of the window. Direct cursor positioning however is relative to the whole screen.

Example: Outline a window

```
  1 width=255
 10 esc$=chr$(27)                          'ESC
 20 cls$=esc$+"E"                          'clear window or screen
 30 window$=esc$+","                       'to set window
 40 nowind$=esc$+"."                       'cancel window
 50 cur$=esc$+"Y"                          'position cursor

100 print cls$;nowind$;                    'clear screen - no window
110 input "top line                        :";top
120 input "bottom line                     :";bot
130 input "left margin                     :";lft
140 input "right margin                    :";rgt
                                           'do not allow 0 coordinate
150 print window$+chr$(31+top)+chr$(31+bot)
                  +chr$(31+lft)+chr$(31+rgt);

200 print esc$+"p";                        'reverse video to emphasise

300 for row=top to bot
310 for col=lft to rgt
320 print cur$+chr$(row+31)+chr$(col+31)" ";        'outline window
330 next col

350 next row

400 print esc$+"q";                        'turn off reverse
410 print esc$+"H";                        'home cursor within window
420 input "press return";a                 'just to pause then do again
430 goto 100
```

The example shows in statement 320 that cursor addressing is relative to the screen and NOT the Window origins. The cursor positioning and the window definition ESCape sequences require that each co-ordinate be offset from 31 decimal or 32 decimal depending upon whether the application is using an origin of 1 or 0 respectively. This example uses an origin of 1, 1.

Note: The last line in this example will scroll up!

Fonts

The ROM BIOS has an in-built character Font limited to the first 128 characters in the standard Ascii set detailed in the appendices. This permits the BOOT sequence to use the screen.

This is an 8 x 8 bit Font, i.e. 8 bytes per character, which drives the screen in 200 line mode.

The RAM BIOS subsequently loads the file "FONT.SYS" into the user RAM which consists of 2 different Fonts for the full 256 Ascii character set. They are:

1. 8 x 8 for 200 line mode
2. 8 x 10 for 256 line mode

The Pointers

00706H	Base of Active Font
0070AH	Length of Active Font
0070CH	Base of Master Font
00710H	Length of Master Font (i.e. of 8 x 8 and 8 x 10)

are then initialised to point to either Font (see Appendix C). The 8 x 10 Font is located at Master Font + 2048 bytes (i.e 256 x 8 bytes).

The character Fonts in FONT.SYS may be altered by using the Edit Font utility supplied with the Apricot F1 or alternatively by modifying the RAM based Font.

The latter may be achieved in two ways:

1. Loading a complete new table and changing the pointers to the active table to reflect the location and length. Refer to Keyboard table example.

   Note that the Master pointers are required by the screen driver to provide for environment switching. Care must be taken in changing them.

2. By modifying one or more locations directly in the existing Font.

Ascii control codes

The Screen Driver handles control codes i.e. codes with an Ascii value which is less than 20 hex as follows:

00 - NUL : no action
01 - SOH : no action
02 - STX : no action
03 - ETX : no action
04 - EOT : no action
05 - ENQ : no action
06 - ACK : no action
07 - BEL : sounds the BELL
08 - BS : moves cursor back one space
09 - TAB : moves cursor right in modulus of eight
0A - LF : moves cursor down 1 line
0B - VT : moves cursor down 1 line
0C - FF : moves cursor down 1 line
0D - CR : moves cursor to beginning of current line
0E - SO : no action
0F - SI : no action
10 - DLE : no action
11 - DC1 : no action
12 - DC2 : no action
13 - DC3 : no action
14 - DC4 : no action
15 - NAK : no action
16 - SYN : no action
17 - ETB : no action
18 - CAN : no action
19 - EMM : no action
1A - SUB : no action
1B - ESC : invokes an ESCape sequence
1C - FS : no action
1D - GS : no action
1E - RS : no action
1F - US : no action

ESCape Sequence Table

The table set out in this section lists the complete set of ESCape sequences for the Generic BIOS.

Appendix B lists the ESCape sequences by type of activity, e.g. WP primitives, Keyboard related, etc.

ESCape sequences consist of a sequence of characters always commencing with the ESCape code 27 decimal. This is best illustrated with BASIC as follows:-

```
PRINT CHR$(27)+"function"+"parameter(s)"
```

where:

function is a Ascii value in the range 20H to 7DH

parameter(s) is an Ascii value in the range 00H to 7FH

(The number of parameters varies according to the function).

ESCape sequences may have no meaning on certain machines. For example those affecting the Microscreen are not implemented throughout the Apricot family. Also certain ESCapes are listed as No-op's. Do not use these ESCape sequences - they are not supported.

The parameters supplied for cursor control sequences are based upon an origin of column 1 line 1. Any other required origin should be relative the base of 32 decimal.

Availability is denoted by the Machine Key in each entry. The Key is:

A - Apricot
F - F1
P - Portable

ESCape sequence table

CHAR	HEX	KEY	Function
#	23		No-OP
$	24	PFA	Transmit Character Sends the character under the cursor into the keyboard buffer.
%	25		No-OP
&	26	PFA	Print Page Outputs the contents of the screen to the line printer. A form-feed is executed first.
'	27	PFA	Print line Outputs the entire line that the cursor is at to a connected line printer. No form feed is executed.
(	28	PFA	Set High intensity Mode Shadow-prints all characters to give the effect of high intensity characters.
)	29	PFA	Set Low intensity Mode Clears the mode set above.
*	2A	A	Change to second character font.
+	2B	A	Clear all high intensity characters.
,	2C	PFA	Set Window size. Takes four parameters in Ascii: <1> Top line + 31 <2> Bottom line + 31 <3> Left-hand column + 31 <4> Right-hand column + 31
-	2D	A	Clear all low-intensity characters.
.	2E	PFA	Reset Window Size. Resets the window size set by code 2C
/	2F	A	Set membrane key LED's.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
0	30	PFA	Sets underline mode. All characters printed have a single line of pixels placed under them to simulate underline.
1	31	PFA	Reset underline mode. Cancels the mode set above.
2	32	A	No-op
3	33	A	No-op

ESCape sequence table (continued)

CHAR	HEX	KEY	Function

4	34	PFA	Change the representation of a key.
			
			This escape sequence takes 3 parameters.
			They are:
			
			<1> - Key mode (ascii)
			1 = normal
			2 = shift
			3 = control
			
			<2> - Key number – 1:
			0 = help... etc.
			(see Appendix B)
			
			<3> - New key character
			Ascii char. or hex equivalent.
			
			`10 PRINT CHR$(27)+"4"+"3"+`
			`CHR$(72)+"C"`
			
			This example changes the key with the legend "C" (downcode number 72), in control mode, to generate an Ascii "C" (43H) as opposed to a binary 03H.
			
			The key has the default attribute of AUTO-REPEAT.
			
			Refer to Appendix B for a list of keys, their corresponding character value, down code and attributes.
			
			Note: The Screen Driver does not accept non-printable characters (range 0 to 31) in ESCape sequences therefore no Key or Key character can be programmed with a value below 32. See <2> and <3> above.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
5	35	PF	Set character foreground colour. This escape sequence takes 1 parameter which is from "0" (30H) to "?" (3FH). This gives 16 possible indexes. For a list of indexes and colours represented by them see escape sequence "]" (5DH). See also the diagram for ESC "6" (36H) below.
6	36	PF	Set block or background colour. This escape sequence sets the colour of all pixels in the character cell that do not make the actual character shape. As above, it takes 1 parameter. Diagram of a character cell: Background (all 0's) 00000000 ——————┘ 00111100 00100100 001001 —Foreground or text 00111100 colour (all 1's) 00100100 00100100 00000000

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
7	37	PF	Set screen environment. This sequence is followed by one of the following mode parameters: "0" - Apricot PC & Xi monochrome compatibility "1" - 80 column full colour display "2" - 40 column mode "3" - 80 column Apricot compatible Display Refer to section Screen environment for operational details.
8	38	PFA	Set literal/Test mode ON The escape sequence tells the screen driver to perform the following action on receiving the next character: Ignore the fact that it is a control code (<20H) and print the character associated with it. This means that the font cell characters under (20H) are printed, rather than obeyed. The ESCape must be sent for each character.
9	39	PFA	Set strikeout mode ON All following characters are displayed with a horizontal line through the centre. This is widely used for deleting data within legal documents.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
:	3A	PFA	Set strikeout mode OFF. The code reverts the action of "9" (39H).
;	3B	PFA	Position cursor to start of status line. The status line is the 25th line of the Screen. The Cursor remains on this line until a position Cursor command is given.
<	3C	A	Display time on MSCREEN.
=	3D		No-OP
>	3E		No-OP
?	3F	A	Enter CALC mode This escape sequence switches on the internal BIOS calculator. It is the same as pressing the "Calc" key.
@	40	PFA	Enter insert mode. After this escape sequence is issued, whenever a character is printed, all the characters to the right of it will be shifted right one place and the character will be inserted in the space created.
A	41	PFA	Cursor UP.
B	42	PFA	Cursor DOWN.
C	43	PFA	Cursor RIGHT.
D	44	PFA	Cursor LEFT.
E	45	PFA	Clear screen. The current window is cleared, and the cursor is homed.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
F	46	PFA	Enter VT52 Graphics mode. VT52 mode displays the VT52 standard graphics characters represented by the Ascii value of Apricot lower case characters.
G	47	PFA	Exit VT52 Graphics mode. Invoke this mode to exit VT52. Failure to do this results in non lower case letters being incorrectly displayed.
H	48	PFA	Home Cursor. The cursor is placed at the top left hand corner of the current text window.
I	49	PFA	Reverse-index and line-feed. This sequence moves the cursor up one line. However if the cursor is at the top of a window then a scroll DOWN of the whole window is performed.
J	4A	PFA	Erase to end of Page. The sequence first of all erases all characters from the cursor position to the end of the current line, and then all subsequent lines below the cursor till the end of the current window or page.
K	4B	PFA	Erase to end of line. This escape sequence erases all characters from the cursor position to the end of the defined right-hand side margin.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
L	4C	PFA	Insert line. This sequence places the cursor to the beginning of the current line, and then inserts one line below the current cursor position by scrolling all subsequent lines down by one place.
M	4D	PFA	Delete line. This sequence places the cursor to the beginning of the current line and scrolls all lines under it up by one place.
N	4E	PFA	Delete character. The character under the cursor is cleared, and all characters to the right are scrolled left by one position. This is active in the defined right-hand margin space.
O	4F	PFA	Exit Insert Mode. This sequence reverses the effect of ESC "@" (40H)
P	50	PFA	Insert single character. This sequence scrolls all characters from the current cursor position to the defined right-hand margin right by one place.
Q	51	PFA	Scroll left. Takes one parameter which is the number of columns plus 31 that the screen is to be scrolled. This is only active in the current window.
R	52	PFA	Scroll right. As above but scrolling left.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
S	53	PFA	Scroll up. As above, but scrolling is up.
T	54	PFA	Scroll down. As above, but scrolling is down.
U	55	A	Enable dual-output to MSCREEN
V	56	A	Disable dual output.
W	57	A	Output text to microscreen only.
X	58		No-OP.
Y	59	PFA	Position Cursor. This sequence takes two parameters. They are the line number and column number in normalised Ascii. eg: `PRINT CHR$(27) "Y" CHR$(10+31) CHR$(15+31)` will position the cursor at line 10, column 15.
Z	5A	PFA	Identify as VT52. This escape sequence is included as most of the screen driver is DEC VT52 compatible. After issuing this sequence the keyboard buffer is filled with three characters which can be read by an application to determine the device type.
[	5B	PFA	ANSI lead-in character. Refer to section ANSI ESCape sequences for details of the ANSI codes supported.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
/	5C	PFA	Place key in keyboard buffer. This sequence takes one parameter which is placed in the keyboard buffer. If the buffer is full the bell will sound and the character will be ignored: e.g. PRINT CHR$(27) "/R" will place an "R" into the keyboard buffer.
]	5D	PF	Set palette code. This escape sequence takes two arguments. The first is the index which needs to be changed, and the second is the colour. e.g. PRINT CHR$(27) "]05" sets index 0 (in this case the background) to colour 5. Refer to the Colour section for full details of index, colour and default palette settings.
^	5E		No-OP
_	5F		No-OP
`	60	PFA	Save environment The first three environment flags are saved. They can then be temporarily changed and restored by another sequence.
a	61	PFA	Restore environment Returns the first three environment flags to their state just after code 60.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
b	62	PFA	Erase from start of page All characters from the top left-hand corner of the current defined display page size to the current cursor position are cleared.
c	63	A	Disable MSCREEN scrolling.
d	64	A	Enable MSCREEN scrolling.
e	65	A	Switch MSCREEN cursor ON.
f	66	A	Switch MSCREEN cursor OFF.
g	67	A	Disable Time and Data display on microscreen.
h	68	PFA	Reverse tab. This sequence has the opposite effect of control code 9 hex, it performs a tabulation operation to the left rather than the right.
i	69		No-OP
j	6A	PFA	Save cursor position The current cursor position is noted within the BIOS.
k	6B	PFA	Restore cursor The cursor is restored to the position it was in before ESC 'j' was executed.
l	6C	PFA	Erase line The line which the cursor is on is cleared. Note that no scrolling takes place
m	6D	PFA	No-OP

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
n	6E	PFA	Return Cursor position. The current cursor position is placed into the keyboard buffer in the ESC+"Y" format with a base column and row of 1.
o	6F	PFA	Erase to start of current line. All characters from the start of the current line up to and including the character under the cursor are cleared.
p	70	PFA	Enter reverse video mode. All characters printed after this sequence are displayed in inverse video.
q	71	PFA	Cancel inverse video mode. Restores the writing mode to the state it was in before ESC 'p' was executed.
r	72	A	MSCREEN echo enable.
s	73	A	MSCREEN echo disable.
t	74		No-OP
u	75		No-OP
v	76	PFA	Wrap at end of line. This escape sequence indicates that the normal screen driver action when the cursor reaches the end of a line should be employed. The action is to return the cursor to the beginning of the next line on the screen.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
w	77	PFA	Discard at end of line. When the cursor reaches the end of the current line, it will remain there and all characters are printed under it.
x	78	PFA	Set environment flags. Takes one parameter:
		PF	G - reset screen palette
		PF	$ - set to Apricot Compatible mode on default screen
			1 - enable line 25
			2 - nothing
			3 - nothing
			4 - nothing
			5 - cursor off
			6 - nothing
			7 - nothing
			8 - set auto LF on receipt of CR
			9 - set auto CR on receipt of LF
			A - nothing
			B - nothing
			C - nothing
		PF	D - smooth screen scrolling
		P	E - LCD contrast up
		PF	F - bell volume down

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
y	79	PFA	Reset environment flags.
			Takes one parameter:
			1 - disable line 25 2 - nothing 3 - nothing 4 - nothing 5 - cursor on 6 - nothing 7 - nothing 8 - no auto LF on receipt of CR 9 - no auto CR on receipt of LF A - nothing B - nothing C - nothing
		PF	D - fast screen scrolling
		P	E - LCD contrast down
		PF	F - bell volume up
z	7A	PFA	Reset all screen drivers.
			Sets all screen drivers to power-on status.
{	7B		No-OP
|	7C		No-OP
]	7D	PF	Reserved for GSX - call is not supported.

Systems interest

Screen Bit Image

The Apricot F1 has a bit screen memory which enables normal text and graphics to be combined on the same screen.

The Bit Image is fully discussed both from a physical and a software point of view in the Hardware section of this manual.

The Screen Driver provides the Display with a choice of environments as discussed in the Applications section.

The block diagram below shows how the Bit Image is logically divided to support the Apricot in the Screen Driver environment.

The Bit Image of 42K bytes is located at address 2000 hex. The Image is divided into 2 Planes. One plane is formed by the low bytes of each word; the second plane by the high bytes. A bit from each Plane is used to derive a choice of 4 colours from the palette.

The Bit Image of each character depends upon the attributes and screen environment and these are discussed fully in the next section.

In text mode the characters from the screen image are coded depending upon the Active character Font into an 8 x 8 or 8 x 10 bit image.

In addition to the Bit Image the Apricot incorporates an area of 200/256 Display Line Pointers located between 001E00 and 00200 hex. Each pointer represents a 640 Pixel line on the Display. Manipulation of the Line Pointers enables very fast scrolling up/down/left/right and in general very sophisticated screen handling.

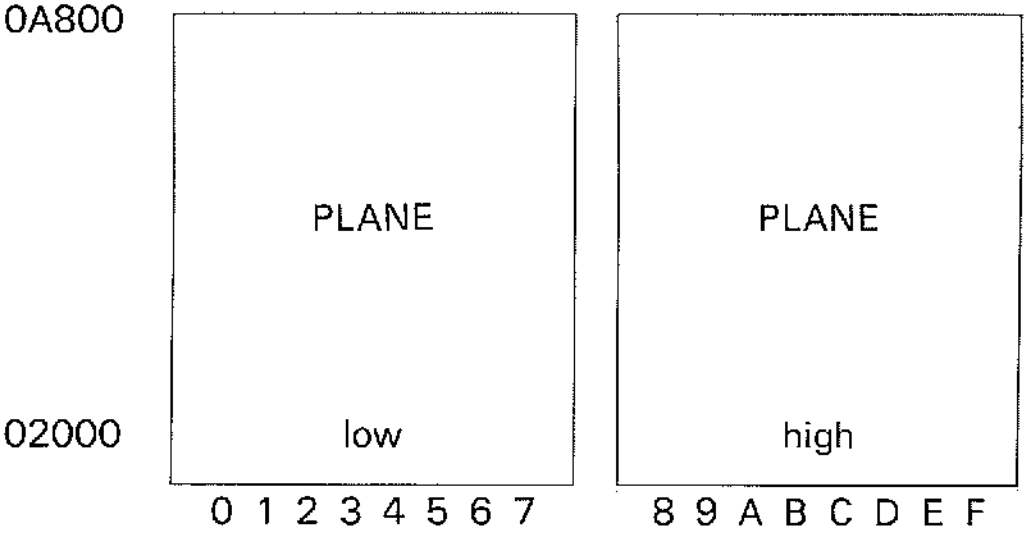
Interface Connection Detail

System Connections

D0 to D7	Data bus. Used to transfer data, commands and status information between the CPU and the FDC.
$\overline{WE}$	Write Enable. Write control input connected to the $\overline{AIOWC}$ control line of the system control bus. Active state, logic low. Used in conjunction with $\overline{CS}$ and the register select inputs (A0, A1) to transfer data and commands from the data bus to the FDC.
$\overline{RE}$	Read Enable. Read control input connected to the $\overline{IORC}$ control line of the system control bus. Active state, logic low. Used in conjunction with $\overline{CS}$ and the register select inputs (A0, A1) to transfer data and status information onto the data bus from the FDC.
$\overline{CS}$	Chip Select. Address input. Active state, logic low. When active, indicates that the FDC is selected for a data/command transfer operation.
A0, A1	Register select lines. Inputs connected to A1 and A2 of the system address bus Used to select internal registers within the FDC for data/command/status transfers, via the data bus as detailed below. A1 A0 0 0 Status/Command registers 0 1 Track register 1 0 Sector register 1 1 Data register
INTRQ	Interrupt Request. Output connected to The NMI request line of the CPU. Active state, logic high. When active, signifies that the FDC has terminated a command operation.

DRQ	Data Request. Output connected to the TEST input of the CPU. Active state logic high. When active, signifies to the CPU that the FDC is ready for a data transfer operation.
$\overline{\text{MR}}$	Master Reset. Input connected to the System Reset control line. Active state, logic low. When active, causes the FDC to reset its' internal registers.
CK	Clock input. 2 MHz clock with a 50% duty cycle for internal timing within the FDC.

Apricot Bit Image RAM



For each word the low to high bits are mapped to form a colour selection for each Pixel, i.e. Bit 0 and Bit 8, Bit 1 and Bit 9 .... Bit 7 and Bit F.

Other options are available though not supported by the Screen driver. 16 colour is derived by mapping the Bit image into 4 planes instead of 2. This can only be achieved in 40 column mode as 4 bits are required to define each Pixel and hence 2 words per character line.

In this case the colour decode hardware maps Bit 0, 1, 8 and 9 to Pixel 0, Bits 2, 3, A and B to Pixel 1 etc. This mode is discussed in detail in the Hardware chapter.

Character attributes

The character attributes are not contained in the Bit Image but located in a separate image as discussed in the section on Screen images.

The attributes differ when using Colour or Monochrome mode as discussed in the section Apricot compatible mode.

The Bit Image is derived from the Display Image by software manipulation. This manipulation differs according to the environment and colour mode.

In Monochrome Apricot Comapatible mode the Bit Image is generated by translating the character via the active Font and then amending the character according to the attributes set by bit manipulation. For example Strikethrough is achieved by "setting" the Pixels in the character centre line of the Bit Image.

In Colour the attributes are interpreted as the foreground and background colour selectors.

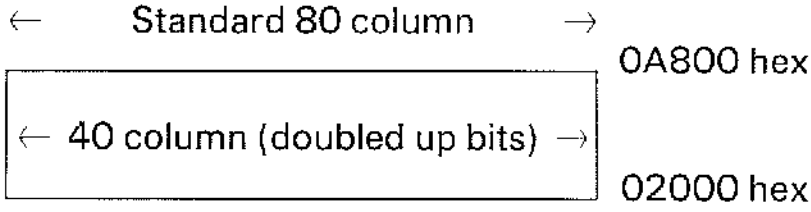
The character is translated via the Active Font and then each bit is mapped into the 2 colour planes to correspond to a selection (index) from the colour palette.

40 Column mode

The 40 column mode is available for TV screen support.

Characters are generated in the same way as described in the previous section but with the difference that they are doubled in width in the Bit Image i.e they occupy a 16 x 8 cell.

Screen image:



Scrolling

Scrolling is achieved through the use of ESCape sequences as described in the Applications sections.

There are four methods of scrolling, i.e.

Left - ESC+"Q"+n
Right - ESC+"R"+n
Up - ESC+"S"+n
Down - ESC+"T"+n

where n is the number of character positions (vertically or horizontally) + 31 Ascii. The complete window is scrolled.

A set of 256 word pointers located at 01E00 hex in the RAM are used to point to a Pixel line in the Bit Image. Scrolling and other effects are achieved by swapping these pointers. For 200 line displays only the first 200 pointers are significant.

Configuration table

The screen driver entry is located at offset 20 hex in the configuration table and contains the following:

0 - line mode	: 0 = 256 1 = 200
1 - environment	: 0 = Apricot compatible monochrome 1 = 4 colour 2 = 40 column
2 - image switch	: 0 = on 1 = off
3 - spare (12 bytes)	

Keyboard driver 3.4

Contents

Overview

Application interest
- Changing the keyboard table
- Implementing STRING keys
- Changing the keyboard driver operation
- Special Keys
- Default STRINGS
- Prefixes
- User Interrupt (F9 hex)

Systems interest
- Initialisation
- Steering
- Down-code handler
- Queues
- Configurator
- Apricot compatibility
- Configuration table

Illustrations

1. Downcodes

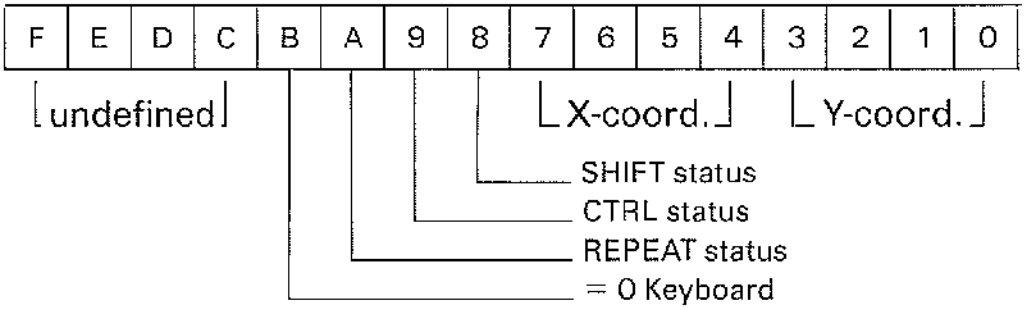
Overview

The Keyboard driver software handles communications between the SIO and MS-DOS. Serial data is received by the SIO from both the Keyboard and the Mouse.

The driver distinguishes between the two and vectors to the appropriate routine. The Mouse driver is of sufficient importance to be treated as a separate section of the drivers. Here we will concentrate on the keyboard.

Applications running under MS-DOS receive data from the keyboard via normal MS-DOS CALL (INT 21H).

The action of pressing a key or combination of keys, e.g. SHIFT+ or CTRL+, will result in the SIO generating an interrupt and passing a packet of data in Hamming coded format to the SIO interrupt handler. The data is then converted and presented to the driver in the AX register in the following format:



A similar packet is sent for the Mouse except that bit B = 1.

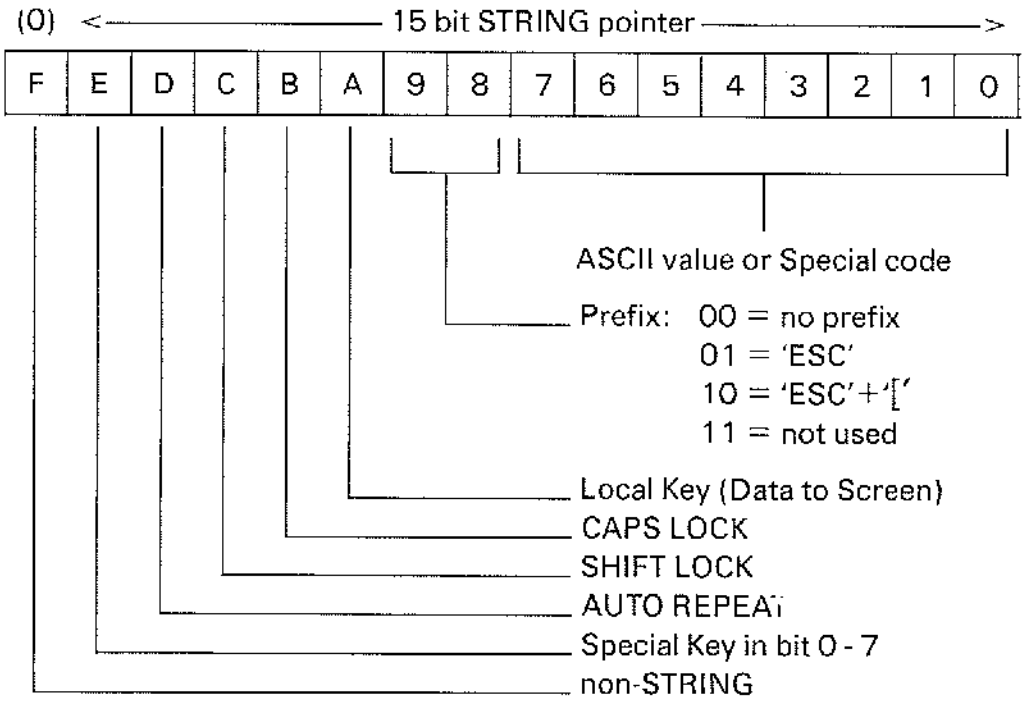
The X and Y coordinates are converted into an offset within the soft keyboard table. The generic keyboard has 104 keys and the offset is in the range 0 to 103. Each of these is referred to as a "down-code".

The keyboard table is divided into sections, each of 104 words, to facilitate the following:

Normal - key without SHIFT or CTRL
Shifted - SHIFT + key
Control - CTRL + key

The attribute bits 8 and 9 of the packet determines which section of the table is to be used to translate the down-code.

The translation of a down-code is dependant upon further attributes in the corresponding entry of the table. Each entry is a word of the following format:



Attribute bits A - F are: 'ON' if set to 1
'OFF' if set to 0.

The exception to this format is when "Non-STRING" attribute bit F is set 'OFF', implying that the down-code is to be translated into a string of data. In this case the format is:

bit 0 - bit E - String offset of 15 bits (i.e. 32K)
bit F - set 0 = STRING

The STRING facility enables any key to be translated into a string of data. e.g. The F1 key may be interpreted in BASIC as the character string RUN plus a CR which results in 4 characters being returned to the application level via MS-DOS.

These STRINGS, which may be of variable length, are appended to the end of the three sections of the Keyboard table described above. The default table is 1024 bytes long - the first 312 words for each of the 3 sections described above, followed by a 200 word table reserved for "Default STRINGS".

The remaining attribute bits have the following meaning:

AUTO REPEAT	When set enables auto repeat on this key.
SHIFT LOCK	When set in the NORMAL entry then the Key is affected by SHIFT LOCK mode. The driver translates the Key from the SHIFTED area of the table.
CAPS LOCK	When set in the NORMAL entry then the Key is affected by CAPS LOCK mode. The driver translates the Key from the SHIFTED area of the table.
LOCAL KEY	When set the translated key is directed to the screen and NOT to the Queue.
PREFIX	The prefix is derived from the value of two bits and dependent upon the value an ESCape sequence is pre-fixed to the translated data (see the section on prefixes).

The default Keyboard Table, if present on the Boot disk, is loaded into RAM at BOOT time. It may subsequently be modified either by replacing it entirely with another table or simply by changing specific entries. This procedure is described in the section Applications Interest.

The Driver translates each key into one or more data bytes which are then appended to a 'ring buffer' queue. If the data will not fit on the end of the queue then it is lost and the BELL is sounded.

MS-DOS accesses the keyboard via the RAM BIOS using calls to the Control Device: typically requests for "get data from queue" or "Flush queue".

Application programs may by-pass MS-DOS and use the same facilities in the Control Device.

This is of advantage where the normal operation of the keyboard driver is not required. For example, if the translation procedure described above is not required, the Application may alter the actions of the keyboard driver to return the down-codes in the queue, without translation. These down-codes are referred to as "Raw data". Various other facilities are also available and detailed in the Control Device chapter.

The following section describes with examples how the applications may take advantage of the Generic keyboard facilities. In addition, it describes in detail all of the Special Keys and their actions.

The Systems interest section describes the modules making up the driver and special features implemented in the software such as Auto-repeat.

Applications Interest

Changing the keyboard table

There are three methods of changing the Keyboard table in RAM, they are:

1. Use the system utilities to modify the system table.
2. Load a new table and change pointers to it.
3. Change values within the default table.

If the modification is to be permanent then the Keyedit Utilities should be used to amend the Keyboard table.

To implement the second option above it is sufficient to load the new table into a reserved area of memory and then to change the Pointer to the "active keyboard table" to reflect the new location. REMEMBER to restore the original pointer if the following Application is not known.

Example: Load a new keyboard table and point to it.

```
100 DIM TABLE%(1023)                          'data area for user Table
110 TABOFF%=VARPTR(TABLE%(0))                 'offset within data segment

120 CALL GETSEG%(SEG%)                        'data segment (refer Appendix F)

200 DEF SEG=0                                 'segment for PEEKS & POKES
210 KS=PEEK(&H0714)+(256*PEEK(&H0715))        'segment - old pointer
220 KO=PEEK(&H0712)+(256*PEEK(&H0713))        'offset - old pointer

240 POKE &H714,(SEG% AND &HFF)                'segment - new pointer
250 POKE &H715,(SEG% AND &HFF00)/256
260 POKE &H712,(TABOFF% AND &HFF)             'offset - new pointer
270 POKE &H713,(TABOFF% AND &HFF00)/256

300 DEF SEG                                   'basic data segment
310 BLOAD "KEYTAB",TABOFF%                    'load new table
```

The memory image of the table in "KEYTAB", which must have been created with a BSAVE statement, is loaded into the array TABLE%. The called subroutine at GETSEG% is defined in Appendix F (Language interfaces to the BIOS). This routine must be merged with the example program for execution purposes.

Statements 210 and 220 simply save the BIOS pointers to the active keyboard table in order that they may be restored after execution of the program.

To change values within the default table simply pick up the Pointer to the base of the table and modify the specific locations offset to the base address.

The example of switching keyboard tables above is dependant upon a table being previously BSAVE'd. Many other methods of loading a Keyboard table may be derived and in certain cases will be necessary. Compiled Microsoft Basic does not support BSAVE, for example, so a different method is required.

The BLOAD and BSAVE method, however, serves the purpose of illustrating the technique and for completeness the example below shows how to save a table.

Example: Save the active keyboard table

```
120 CALL GETSEG%(SEG%)                          'data segment (refer Appendix F)

200 DEF SEG=0                                   'segment for PEEKS & POKES
210 KS=PEEK(&H0714)+(256*PEEK(&H0715))          'segment - old pointer
220 KO=PEEK(&H0712)+(256*PEEK(&H0713))          'offset - old pointer
230 KL=PEEK(&H0716)+(256*PEEK(&H0717))          'length

300 BSAVE "KEYTAB",KO,KL                        'save active table
```

This example may be brought into perspective by combining it with the previous example and the one that follows in the next section. The result is a Keyboard Table being saved with a few changes from the default. Then the modified table may be loaded at any time as described above.

Minor changes to the Keyboard Table at run time may also be made by simply picking up the Pointer to the base of the table and modify the specific locations offset to the base address.

Alternatively further techniques are described in the Screen Driver chapter in the ESCape sequence section.

Implementing Key STRINGS

There are many cases in practice where the depression of a single key can eliminate the tedium of typing in a complete sequence of characters. In BASIC certain verbs are commonly used in the interactive mode while testing, for example:

PRINT - to check contents of variables
LOAD - to load files
SAVE - to save files

Whatever the application, keys may be modified to reflect the input of a complete string of characters to the keyboard. Typically the function keys would be used to accomplish this.

The Key STRINGS are located at the end of the 3 sections of the down-code translation table. In order to add Key strings to the list it is neccessary to perform the following:

1. Insert the new Key STRING and attributes at the end of any existing assignments.
2. Modify the down-code table to point to it.

The following program illustrates how to modify the 9 function keys F1 - F9 to represent BASIC commands as follows:

F1 - RUN	F6 - PRINT
F2 - AUTO	F7 - LOAD
F3 - LIST	F8 - SAVE
F4 - LLIST	F9 - DELETE
F5 - RENUM	

In addition F1 to F5 are set up to generate the STRING + a carriage return when SHIFT+Fkey are pressed.

The program sets up the following parameters in DATA statements:

MODE - 0 = normal, 1 = SHIFT, 2 = CONTROL
KEY - down-code
KEY$ - STRING
CR - 0 = no CR, 1 = with CR

It searches from the end of the table for the first free STRING entry and then places each new entry successively at the end of the table and updates the pointer in the relevant part of the down-code table. If the CR parameter is non-zero then an additional entry is made in the SHIFT section with the addition of a CR (Carriage return).

Example: Implementing Key Strings

```
100 REM set up function keys
110 DATA 9 'no of items in list
120 DATA 0,0,"RUN ",1
130 DATA 0,1,"AUTO ",1
140 DATA 0,2,"LIST ",1
150 DATA 0,3,"LLIST ",1
160 DATA 0,96,"RENUM ",1
170 DATA 0,4,"PRINT ",0
180 DATA 0,5,"LOAD ",0
190 DATA 0,6,"SAVE ",0
200 DATA 0,7,"DELETE ",0

500 REM get pointers
510 DEF SEG=0
520 KS=PEEK(&H0714)+(256*PEEK(&H0715)) 'segment
530 KO=PEEK(&H0712)+(256*PEEK(&H0713)) 'offset
540 KL=PEEK(&H0716)+(256*PEEK(&H0717)) 'length

600 DEF SEG=KS
610 J=KO+KL-1                          'end of table
620 WHILE PEEK(J)=0 : J=J-1 : WEND
630 J=J+1 'start of free string table
635 REM "check for sufficient space !!!"

640 READ N 'no of DATA statements
650 FOR I=1 TO N : READ MODE,KEY,KEY$,CR

660 POKE J,LEN(KEY$) 'STRING length
670 POKE J+1,0        '        attrib
680 FOR X=1 TO LEN(KEY$)
690 POKE J+1+X,ASC(MID$(KEY$,X))
700 NEXT X

720 DEF SEG 'BASIC data
730 P%=J 'set up 15 bit pointer
740 P1=PEEK(VARPTR(P%)):P2=PEEK(VARPTR(P%)+1)
750 DEF SEG=KS 'put it in down-code table
760 POKE KO+(MODE*208)+(KEY*2),P1
770 POKE KO+(MODE*208)+(KEY*2)+1,P2

780 J=J+2+LEN(KEY$)+1 'point to next free entry
790 IF CR <>0 THEN KEY$=KEY$+CHR$(13): MODE=CR: CR=0 : GOTO 660
800 NEXT N
```

Note: The example may modify the default table settings which already have pre-defined functions such as invoking the calculator. Refer to the Keyboard Table in the Appendix for details.

Changing the Keyboard driver operation

The Overview of this chapter describes how the Keyboard driver works by default.

The mode of operation may be changed by "special" keys, which are detailed in a later section, or by calls to the Control device. The latter is considered below.

The Keyboard driver normally translates down-codes into data from the active keyboard table and then places the data in the queue.

Some applications, however, require that the "raw" down-codes are not translated. This may be achieved by changing the "Fall-thru" mode via command 0004H of the Control device.

Example: The "Fall-thru" mode

```
100 DEF SEG=60H : IO=0                          'point to Control device vector
105 COM%=7:DAT%=0:GOSUB 1000                    'set HELP ignore off
110 COM%=4:GOSUB 2000                           'toggle the "Fall-thru" switch
120 IF RET%=1 THEN PRINT "RAW Down-codes:"
130 IF RET%=0 THEN PRINT "ASCII normal values"
200 COM%=&HB:gosub 1000                         'wait for char. in queue
210 PRINT RET%;                                 'print its value
220 IF RET%=1 THEN 110                          'if true then toggle mode
230 GOTO 200
1000 DEV%=&H32
1010 CALL IO(DEV%,COM%,DAT%,RET%)               'execute command
1020 RETURN
2000 DAT%=-1: GOSUB 1000                        'get current setting
2010 DAT%=RET% XOR 1: GOSUB 1000                'toggle it
2020 RETURN
```

The example displays the current "Fall-thru" setting and waits for a key to be pressed which results in data being placed in the queue.

The options are to display the down-code in the range 1 - 104 or to display the translated value of the key from one of the three sections of the table, i.e. Normal, SHIFT or CTRL.

The program toggles the "Fall-thru" mode whenever it receives the value of 1 in RET%. This value is the result of different key depressions in the two modes, they are:

HELP (or F1) - in "Fall-thru"
CTRL+A - in normal mode

The HELP key has special significance to the driver and if it were detected in this program then it would not work. For this reason the HELP ignore call is made at the beginning of the program. Refer to Systems interest for further details.

The subroutine at statement 2000 illustrates the GET/SET feature of the Control Device. The setting of invalid data in line 2000 results in the call returning the current setting unchanged. Line 2010 then toggles the setting with an exclusive OR and calls the Control Device again.

Refer to the Control Device for further calls affecting Keyboard driver operation.

Notes:

1. The down-codes presented to the driver are in fact in the range 0 to 103. However the driver increments "raw" data to generate a range 1 to 104 before placing it in the queue.
2. In the ASCII mode, keys which are designated as "STRINGS" will be actioned and not displayed. This means that the corresponding string will be placed in the queue.

Special Keys

Certain keys in the down-code keyboard table are designated special keys by setting attribute bit E hex to 1. In these cases bits 0 - 7 specify the key type. These types have a specific meaning to the Keyboard driver i.e. they act as switches. Each type by default is associated with a key in one or more of the normal, SHIFT and CTRL. The types and their respective meanings are as follows:

00 reserved for spare keys - the driver takes no action when these are encountered.

01 this type toggles the SHIFT LOCK and CAPS LOCK driver modes and their respective LED's. i.e.

- if in CAPS LOCK then clear lock and LED
- if in SHIFT LOCK then clear lock and LED
- if in CONTROL mode set SHIFT LOCK and LED
- if in NORMAL mode set CAPS LOCK and LED

02 no action on Apricot F1 and Portable

03 no action on Apricot F1 and Portable

04 no action on Apricot F1 and Portable

05 sets/resets the STOP mode and LED

06 sets/resets CALCULATOR mode and sends start/stop sequences to CALCULATOR.

07 this generates an F7 hex VOICE interrupt.

Refer to the default Keyboard table in Appendix B for down-code designations of these keys. Under normal circumstances these should not be changed.

The generic control device may also be used to invoke these functions via software control. Refer to the Control Device Chapter, Keyboard driver section and the command "Get/Set Keyboard status".

Default STRINGS

The default keyboard table has a number of pre-defined STRINGS. These are invoked by special key combinations. The STRINGS are located immediately after the end of the 3 sections of the down-code table.

These strings may be freely changed or re-designated as required by the application. Note that the Example program to add STRINGS to the list in an earlier section searched from the end of the allocated table space until it found the end of these default STRINGS. It does not need to know exactly where or what their size is.

The Default strings have been implemented to provide extra facilities for the Keyboard, and the colour monitor.

The keys and their functions for the Portable are:

CTRL + F1	- 80 column Apricot compatible mode on the LCD
CTRL + F2	- 80 column multi-colour
CTRL + F3	- 40 column mode on the LCD
CTRL + F4	- 80 column Apricot compatible on the Display
CTRL + UP arrow	- LCD contrast up
CTRL + DN arrow	- LCD contrast down
CTRL + R arrow	- Bell volume up
CTRL + L arrow	- Bell volume down
SHIFT + UP arrow	- Fast screen scrolling
SHIFT + DN arrow	- Smooth screen scrolling

The keys and their functions on the F1 are:

CTRL + F1	- 80 column Apricot compatible mode
CTRL + F2	- 80 column multi-colour
CTRL + F3	- 40 column mode
CTRL + F4	- as CTRL + F1
CTRL + R arrow	- Bell volume up
CTRL + L arrow	- Bell volume down
SHIFT + UP arrow	- Fast screen scrolling
SHIFT + DN arrow	- Smooth screen scrolling

Prefixes

Each entry in the down-code table has a facility to prefix the data value of the key with an ESCape sequence. The two bit prefix in the attributes has the following meaning:

00 - no prefix

01 - prefix with 'ESC' this facility is used with screen control keys such as HOME, CLEAR and the arrow keys. The character 1B hex (27 decimal) is prefixed to a data character and placed in the queue.

10 - prefix with 'ESC'+'['

'ESC' and '[' prefix an ANSI sequence.

11 - is not used

Refer to the Keyboard Table in the Appendix for more details.

User Interrupt (F9 hex)

The Keyboard User Interrupt (F9 hex) is invoked by the driver prior to placing the decoded key character(s) in the Receive Queue.

A copy of the data in the AX and BX register is passed to the user interrupt with the following format:

Low byte = Received character (translated)

High byte = Count of the number of chars. This will be 1 for non-STRING keys. For STRING keys it will be a count of the number of characters to expect inclusive of the current low byte.

The User routine, unless otherwise initialised by the Application, is a dummy routine which will return BX unchanged and AX = 0FFFF hex.

Applications may define a User interrupt to replace the dummy routine. In particular this is useful for the handling of "raw" down code data.

If the routine returns AX = 0FFFF hex then the Driver places the contents of register BL in the Queue.

If the routine returns AX = 0000 hex then the character is ignored, i.e. thrown away.

Any translation or manipulation of the characters may take place and be returned to the Keyboard driver in the BL register for subsequent addition to the queue.

Systems interest

The Keyboard driver consists of various modules which include the following functions:

1. Initialisation
2. Steering

   clock set-up
   time and date entry
   raw code handling
3. Down-code handling
4. Queue handling
5. Configuration

The drivers are not directly accessible by applications software. The correct use of them should always be via the Control device.

In the following sections, each module is discussed to give an overview of internal operation and to provide more detail about certain Control device calls.

Initialisation

This module is executed by the Boot process and performs the following:

zeroises and initialises queue

initialises SIO and keeps copy

Note: The Control device call to initialise only initialises the queue.

Steering

This module performs the function of receiving the data from the SIO and filtering it to the appropriate routine. The following steps are taken:

1. Is the data a Mouse packet?
   Execute the Mouse interrupt 3. This interrupt handles the Mouse packet completely.
2. SET TIME key pressed?
   If the data is the SET TIME key then the 25th line of the display is enabled. The Time/Date prompt is displayed and the next 10 characters of input are assumed to be in the format HH MM DD MM YY (with no spaces in between). The clock within the keyboard is set to this time.
3. TIME/DATE key pressed?
   This key resets the clock implemented by the clock driver with the current setting of the keyboard clock.
4. KB LOCK pressed?
   This key enables the user to lock the keyboard to prevent accidental access. The key has a simple toggle action; lock or., lock off. The driver also controls the corresponding LED on the front panel according to the LOCK status.
5. Normal XY keycode?
   If the LOCK is on then ignore any key. If the XY code is 80 hex or greater then it is ignored. The XY code is translated into a down-code in the range 0 to 103. The routine to handle normal keyboard codes is called.
6. RAW mode active?
   If Fall-thru mode is on then the down-code is adjusted to the range 1 to 104 and added to the queue.
7. HELP key pressed?
   If in RAW mode and the HELP ignore status is 'off' then toggle reset the RAW mode off.

Down-code handler

This is the principle module of the Keyboard driver. It is broken down into a number of sections.

The down-code is passed to the handler in the AX register with the status in the high byte and the down-code itself (with a value derived from the XY coordinates in the range 0 to 103), in the low byte.

Firstly the down-code is used as an offset into one of the three sections of the active Keyboard table to produce one of the following words:

DATA	where bit 0 - 7	= ASCII key data
	bit 8 - 9	= prefix 00 none
		01 'ESC'
		10 'ESC'+'['
		11 'ESC'+'O'
	bit A	= Local
	bit B	= CAPS LOCK
	bit C	= SHIFT LOCK
	bit D	= AUTO REPEAT
	bit E	= 0 - not SPECIAL
	bit F	= 1 - not STRING
SPECIAL	where bit 0 - 7	= type of special key
	bit 8 - 9	= 00 - no prefix
	bit A - D	= 0000 - ignored
	bit E	= 1 - SPECIAL
	bit F	= 1 - Non-string
STRING	where bit 0 - E	= 15 bit string pointer
	bit F	= 0

If the CAPS/SHIFT LOCK mode is set on then down-codes received in NORMAL mode whose corresponding entry has CAPS/SHIFT LOCK enabled are vectored to the SHIFTed area of the table.

Refer to the overview in this chapter for a definition of the bit settings within each word.

The following sequence then takes place within the down-code handler.

1. SPECIAL keys are filtered and processed to affect keyboard status and LEDS as follows:

CAPS LOCK	if in CAPS LOCK clear lock and LED if in SHIFT LOCK clear lock and LED if in CTRL mode set SHIFT LOCK/LED if normal mode set CAPS LOCK/LED
STOP	sets/resets keyboard status and LED
CALC	sets/resets keyboard status sends start/stop sequences to calculator
VOICE	executes an F7 hex software interrupt

2. STRING and DATA keys are filtered and sent to the destination prefixed as necessary. The only difference being that the for DATA only one character is sent.

   The 15 bit string pointer is to a location in the table with the following packet of data in bytes:

   Length (in bytes 'n')

   Type (as in normal entry bits 8-15)

   'n' DATA bytes

If the Local bit is set then the data is sent directly to the screen driver. If the screen driver is not active then the data is ignored.

Queues

The Keyboard I/O queue is an 80 byte circular buffer.

The Keyboard down-code handler passes the data generated by each relevant key to the Queue handler.

The Queue handler performs an interrupt F9 hex, which is described in the section on User interrupts on a previous page.

If there is insufficient room in the Queue then the BELL is sounded and the data is ignored.

The Queue may be accessed directly through the Control device.

The module consists of three distinct areas, they are:

1. Queue filler which adds one or more characters to the Queue. This area is called by the down-code handler and may also be called via commands 8 and D hex of the Control device.
2. The "look-ahead" facility which will return the next output character from the Queue without updating the I/O pointers. Command C hex.
3. The Queue reader which returns one or more characters from the Queue. Commands B hex and E hex.

Configurator

This module facilitates modification of the Keyboard Status and hence the way in which the Driver operates.

The Status may be altered directly by the Keyboard driver on receipt of certain keys or it may be programmed via the Control Device command F hex.

The Status byte has the following settings (bit = 0, is OFF, bit = 1 is ON):

bit 0 = Ignore HELP key in Fall-thru mode
bit 1 = STOP
bit 2 = CALC
bit 3 = SHIFT LOCK
bit 4 = Fall-thru
bit 5 = VOICE LED (On Portable only)
bit 6 = LOCK
bit 7 = CAPS LOCK

The Control Device may be used to both interrogate the Status and set it, simultaneously if necessary, by ANDing the byte with a specific mask to determine the current settings (typically with FF hex to return all status bits) and ORing with another byte to reset Status bits

Apricot compatibility

The Keyboard table format, function and handling is exactly the same on the generic range of Apricot Hardware.

The default size of the table is 1024 bytes divided into 3 sections, i.e. Normal, Shifted and CTRL; each of 104 words with the remaining space being reserved for string keys.

The key numbers are the same with the following qualifications:

1. The function keys F1 to F4, F6 to F10 are mapped onto the original Apricot pc/xi fixed function keys HELP ... FINISH (i.e. by legend rather than number. In addition the 97th entry is for F5 (VOICE).
2. In the default keyboard tables the function keys produce the following codes:

Normal	F1 (HELP)	= 177
	F2 (UNDO)	= 178
	F3 (REPEAT)	= 179
	F4 (CALC)	= 180
	F5 (VOICE)	= 185
	F6 (PRINT)	= 181
	F7 (INT)	= 182
	F8 (MENU)	= 183
	F9 (FINISH)	= 184
SHIFT:	F4 (CALC)	activates the calculator
	F5 (VOICE)	activates VOICE on the Portable
	F6 (PRINT)	activates hardcopy
CTRL	F1 (HELP)	80 Col Apricot compatible (LCD on the Portable)
	F2 (UNDO)	80 Col colour
	F3 (REPEAT)	40 Col mode
	F4 (CALC)	80 Col Apricot compatible (Display on the Portable)

The Membrane Keys (MicroScreen function keys) and their respective entries in the table are not used.

The special keys TIME/DATE, SET TIME, KEYBOARD LOCK, REPEAT RATE & RESET do not appear in the table. These are filtered out from the XY codes passed from the keyboard.

Certain "Special Keys" are no longer supported.

They are:

02 - right hand shift
03 - left hand shift
04 - Control Key
07 - Microscreen echo toggle

One additional "Special Key" is supported, i.e.

08 - VOICE (generates interrupt F7 hex)

The default string table for the Apricot F1/Portable is detailed in the section "Default strings".

Despite these differences the standard Apricot Keyboard Table may be used on the Portable and F1.

Configuration table

The section of the Configuration table reserved for the Keyboard Driver is located at displacement 10 hex in the table. It contains the following

Offset	Function
00	Key click volume (range 0 to F hex) 0 = full/F = off
01	Auto-repeat master enabler 0 = off/1 = on
02	Auto-repeat lead-in delay 1 to 255 times 20 ms
03	Auto-repeat interval of 1 to 255 times 20 ms
04	Microscreen mode (0 = time & date) (1 = screen echo)
05 - 0F	spare

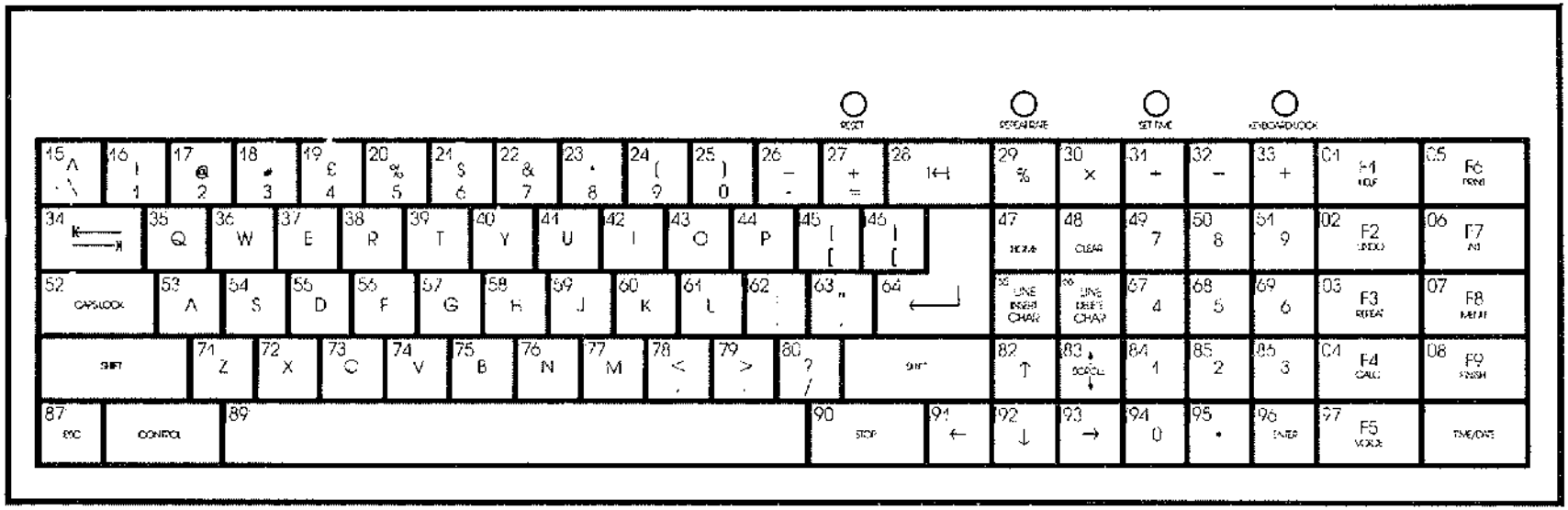


Figure 1. Downcodes

Serial I/O Driver 3.5

Contents

Overview

Applications interest
- Configuring the Serial Driver
- Generic differences
- User Interrupts

Systems interest
- Configuration table

Overview

The Apricot Generic Serial I/O Driver provides asynchronous communications support for the following devices:

RS-232
MOUSE

The Control Device interface provides facilities to configure the Driver for different transmission speeds, line protocols etc.

Further, a call to the Control Device enables the Application, in conjunction with User interrupt (F0 hex), to switch control of the RS-232 channel from the BIOS to itself. This is described in the section on User interrupts.

The I/O port addresses and structure vary throughout the Apricot range but this is transparent to Applications using the Control Device. For Applications which access the hardware directly the above call to the Control Device provides details of the addresses and structure.

The RS-232 channel is switchable in the driver between Serial Mouse and RS-232 communications.

The Serial Mouse, when enabled, is handled by User Interrupt (FA hex). Details are described in the Mouse driver booklet.

The Applications section of this chapter details how to configure the Serial I/O Driver to the required Device and operational environment. The Systems interest section provides details of the configuration data governing the Serial Driver.

In RS-232 mode the Serial Driver may be configured through the Control Device calls to provide the following application environments:

Exclusive User interrupt control (F0 hex)

BIOS control which includes:
XON/XOFF protocol
RTS/CTS protocol
DTR/DSR protocol

Applications which require transparent (i.e non-BIOS) handling of the SIO are considered in the section "User Interrupts". Other applications, which are under BIOS control are discussed in the next section Configuring the Serial Driver.

Applications interest

Configuring the Serial Driver

The calls to the Control Device provide the Application with comprehensive facilities to configure the SIO serial interface in terms of protocol, transmission speeds, data format, etc., for asynchronous communications.

They also provide direct access to the control and status lines of the SIO to enable the application to use any variation, as required, of the standard line protocols provided. The requirement for such direct access is inherent in the field of communications. So many variations occur in external devices and their protocol as to exclude them from the scope of this chapter.

A common requirement in this field however is support of a Serial printer. The Apricot generic BIOS is usually configured to default printer output to the parallel Centronics interface. The example given below demonstrates how to configure the BIOS and switch to Serial printer communications.

The BIOS may be configured to Boot with the printer output directed to the Serial port. The Label Sector must be set to specify a Serial printer and the configuration data table must be set to the required transmission characteristics of the printer.

Alternatively in order to switch an existing Application during Run time to use a Serial printer, the Serial Driver must be configured to the requirements of the individual printer using calls to the Control Device and then a further call to the Control Device to disable parallel support and enable serial.

The example provided below illustrates this procedure for a printer with XON/XOFF protocol and the following option settings:

Switchable XMIT/RCVE baud rates and data bits
Stop bits (1, 1.5 or 2)
Parity (none,odd, or even)

Example: Configuring a Serial printer

```
 10 CLS$=CHR$(27)+"E"    'clear screen escape sequence
 20 DATA "NONE","ODD","EVEN"
 30 FOR I=1 to 3: READ PTY$(I): NEXT I
 40 DATA 1,1.5,2
 50 FOR I= 1 TO 3 : READ STOB(I) : NEXT I
 60 DATA 50,75,110,134.5,150
 70 DATA 300,600,1200,1800,2400
 80 DATA 3600,4800,7200,9600,19200
 90 DIM BAUD(15): FOR I=1 TO 15 : READ BAUD(I) : NEXT I
100 PRINT CLS$;"General data:"
110 INPUT " stop bits (1-3): "; RTS%
120 INPUT " parity (0-2): "; RTP%
130 PRINT

170 PRINT "XMIT parameters: "
180 INPUT " baud rate (1-15)  : "; TXB%
190 INPUT " data bits (5-8): "; TXD%
200 PRINT

240 PRINT "RCVE parameters:"
250 INPUT " baud rate (1-15)  : "; RXB%
260 INPUT " data bits (5-8): "; RXD%

300 DEV%=&H34                                  'Serial I/O Driver
310 COM%=4:DAT%=TXB%:GOSUB 900                 'set xmit baud
320 COM%=5:DAT%=RXB%:GOSUB 900                 'set rcve baud
330 COM%=6:DAT%=TXD%:GOSUB 900                 'set xmit data bits
340 COM%=7:DAT%=RXD%:GOSUB 900                 'set rcve data bits
350 COM%=8:DAT%=RTS%:GOSUB 900                 'set stop bits
360 COM%=9:DAT%=RTP%:GOSUB 900                 'no parity
370 COM%=&HA:DAT%=1: GOSUB 900                 'enable xon/xoff xmit
380 COM%=3:GOSUB 900                           'now set Serial Driver
390 DEV%=&H35:COM%=6:DAT%=1:GOSUB 900          'select serial printer
400 LPRINT "Serial printer test: "
410 LPRINT
420 LPRINT "xmit baud rate " BAUD(TXB%)
430 LPRINT "rcve baud rate " BAUD(RXB%)
440 LPRINT "xmit data bits "  TXD%
450 LPRINT "rcve data bits "  RXD%
460 LPRINT "stop bits" STOB(RTS%)
470 LPRINT "parity" PTY$(RTP%+1)
490 END

900 DEF SEG=&H60:I0=0:RET%=0:CALL I0(DEV%,COM%,DAT%,RET%):
    RETURN
```

Reference to the Control Device chapter will provide details of the calls used in the program.

It is important to note the order of the call in statement 380. All the previous calls simply set up the configuration table which is detailed later in this chapter. Statement 380 actually re-programs the Serial Driver and the SIO.

Printer output is assumed to be directed by default to the Parallel Device driver. Statement 390 switches output to the Serial Device driver and updates the Configuration table.

All printer output within the program is then directed to the Serial Device. Provided that no other commands are given this condition will remain on exiting the program and BASIC. This means that all other output from applications through MS-DOS will also be directed to the Serial port.

This is easily demonstrated by use of the CTRL P command in MS-DOS. i.e.

A>CTRL P

A>DIR

will direct all screen output to the currently selected printer device.

A>CTRL N will cancel the CTRL P

It is important to note that the changes to the Configuration table in RAM will remain active until a cold BOOT is invoked. Applications which require to re-instate the Driver to it's former condition should temporarily store the Serial Driver configuration and then restore it as necessary.

Further, the XON/XOFF codes as defined in the Configuration table may not be compatible with the external device. If this is so then these may be altered within the configuration table. The procedure to achieve both of the above points is detailed in the section below on the Configuration table.

Statement 370 enables the XON/XOFF control of the transmit routines within the driver, i.e. receipt of XON/XOFF characters from the external device will restart/suspend the transmission procedure.

Generic differences

The Apricot generic Serial Driver provides the same support on the F1 and the Portable with the exception of a limitation in the F1 transmission rates.

The F1 does not support split rates of transmit/receive. Care must be taken in ensuring that the transmit/receive baud rate are specified equal.

In addition the F1 does not support 3600, 7200 and 19200 baud. Any attempt to specify one of these rates results in the next lower rate being used, i.e. 2400, 4800 and 9600 respectively.

The Hardware I/O addresses and port structures are different in the F1 and Portable, however this is catered for automatically by using the Control Device.

A Control Device call provides the application with full details of the Hardware configuration. This is detailed in the next section.

User Interrupts

The Serial Device driver can be switched between:

BIOS control, and

User interrupt control (Int F0 hex).

The use of User interrupt control is limited to applications capable of accessing the 8086 and Z80 SIO registers. Details given below therefore relate only to machine code level. (For low level details of the Z80 SIO, refer to the Serial Interface chapter in the Hardware section).

The selection of User Interrupt control is achieved by the Control Device call 1A hex "Set/Reset external SIO control".

This call returns the SIO I/O addresses and structure in the PX parameter (AX register).

AL is the base I/O port address, which relates to the following offsets:

[base + 0] = Channel A data

[base + 2] = Channel A status

[base + 4] = Channel B data

[base + 6] = Channel B status

AH defines the RS-232 channel:

0 = Channel A
1 = Channel B

The hardware chapters provide a detailed description of the SIO registers and how to access them.

Subsequent interrupts on the RS-232 channel are filtered and the interrupt F0 hex is invoked with the AX register containing "000x" which is a vector to one of the 8 conditions as given below.

For the F1 the possible settings are:

0 = Ch B TX buffer empty
2 = Ch B External Status int
4 = Ch B RX ready
6 = Ch B Special receive

For the Portable the possible settings are:

8 = Ch A TX buffer empty
10 = Ch A External Status int
12 = Ch A RX ready
14 = Ch A Special receive

The user interrupt routine handles the call as required. The nature of the interrupt is given below:

TX buffer empty - SIO ready for next character

External/Status interrupt - Line interrupt CTS/DSR etc

RX ready - SIO has received a character

Special Receive - Error in device receive channel.

Configuration table

The block entry for the Serial Device driver begins at location 0030 hex within the configuration table and is of the following format:

Offset	Function	Comment
0000	TX baud rate.	Range 1 - 15 dec. Note 1.
0001	RX baud rate.	Range 1 - 15 dec. Note 1.
0002	TX data bits.	Range 5 - 8.
0003	RX data bits.	Range 5 - 8
0004	Stop bits.	1=1, 2=1.5, 3=2.
0005	Parity check.	0 = no check, 1 = check. Note 2.
0006	Parity type.	0 = none, 1 = odd, 2 = even.
0007	TX XON/XOFF protocol.	0 = off/1 = on.
0008	RX XON/XOFF protocol.	0 = off/1 = on.
0009	XON character code.	Default DC1 (11 hex).
000A	XOFF character code.	Default DC3 (13 hex).
000B	XON/XOFF RX buffer limit.	Note 3.
000D	DTR/DSR protocol.	0 = off/1 = on.
000E	CTS/RTS protocol.	0 = off/1 = on.
000F	Number of nulls after CR.	
0010	Number of nulls x 10 after FF.	
0011	Auto LF after CR.	0 = off/1 = on.
0012	reserved.	

Notes:

1. Refer to Control Device.
2. The Parity check entry is not used.
3. The number of bytes before end of RX buffer. When the buffer fills to this point then an XOFF is transmitted. When it drops below this point then XON is transmitted.

The entries are modified from time to time by calls to the Control Device as can be seen in the example above.

If the application requires this table to be switched back to its original BOOT state following calls to the Control Device, it is essential that a copy is taken before modification. This is best achieved by making dummy calls to the Control Device GET/SET functions which return the current settings.

The Configuration table may be accessed freely but should only be altered with the generic Control device calls. One exception is the modification of the XON/XOFF characters. In certain cases the external device may send different character codes to represent XON/XOFF. The Application program must cater for these instances. An example of how to locate the configuration table and modify this entry is given below.

Example: Changing the XON/XOFF characters

```
100 DEF SEG=&H70                    'segment for Config pointer
120 CS=PEEK(2)+(256*PEEK(3))        'Config segment
130 CO=PEEK(0)+(256*PEEK(1))        ' offset
140 DEF SEG=CS                      'define segment
150 XON=PEEK(CO+&H39)               'pick up default XON
160 XOFF=PEEK(CO+&H3A)              'XOFF
170 POKE CO+&H39,&H1                'new XON=1
180 POKE CO+&H3A,&H3                'new XOFF=3
```

The example picks up the pointer to the Configuration Table from the pointer area in RAM and saves the existing values. (Statements 120 -160). A new character for both XON and XOFF is then placed in the table. (Statements 170 and 180).

The Serial Driver, if invoked for XON/XOFF protocol, will subsequently use these two characters to filter off the devices XON/XOFF characters.

Parallel I/O Driver 3.6

Overview

The Parallel I/O Driver provides support for the Centronics output port.

Applications Interest

Facilities are provided through the Control Device for MS-DOS and Application use.

The Driver supports the BUSY line only.

After Booting the system printer output is directed to the default device (either Parallel or Serial) as specified in the configuration data in the Label Sector.

A switch is available within Control Device calls to toggle between Parallel and Serial.

A 2K byte buffer is used by the Parallel Driver. If printer output is switched to Serial then the Parallel buffer is appended to the 512 byte Serial Auxiliary transmit buffer.

The Control Device also provides a call for Auto LF after CR setting.

Systems Interest

The following data is held at offset 50 hex in the configuration table:

Offset	Function
0000	auto LF after CR (0 = off, 1 = on)
0001	not used
0002	not used
0003	not used
0004	BIOS error report (0 = off, 1 = on)
0005	11 spare bytes reserved

Clock driver 3.7

Overview

The Clock Driver provides the following facilities:

An internal date/time clock

A scheduler for use by peripheral drivers

A User interrupt (FF hex)

The BIOS support via the Control Device consists of three calls, they are:

Initialise, which resets the date/time clock to 00:00:00 on the 1st January 1980.

Read the date/time

Set the date/time.

The User interrupt FF hex provides Applications with a timing facility if required.

Applications interest

The Clock Driver generates a Clock Interrupt (User interrupt FF hex) every 20ms.

Applications which require a timing facility must provide an interrupt routine and modify the Software interrupt vectors to point to it.

Refer to the Guide to the BIOS chapter, section Software interrupts, for details of how to install a User interrupt routine.

In addition the following rules govern the use of the Clock interrupt:

The routine must not last more than 10ms.

The User routine must ensure that the SS, DS and ES registers are preserved.

The routine must not change the Stack unless it disables interrupts.

Calls to either the BIOS or MS-DOS are not allowed.

Systems interest

The Clock Driver maintains a System Date/Time clock relative to midnight on the 1st January 1980 to an accuracy of 2 hundredths of a second.

In addition it provides certain internal facilities for peripherals. These are:

Pre-Boot arrow flash and countdown

Sound driver timeout

Floppy and Winchester head select and movement timing

Printer character output timeout

On Booting the Apricot, a timeout is performed for 10 secs to enable the operator to change the setting of the keyboards internal battery powered clock.

At the end of the timeout automatic Booting takes place if a Bootable disk is present. In this case the System clock is not updated and reflects the time from the initialise default as stated above.

When the TIME/DATE key is pressed the System Clock is always updated from the Keyboard clock.

If the machine is still in the pre-Boot stage then an attempt to Boot takes place.

The internal keyboard clock may be changed at any time by pressing the "Set Time" button and then entering a new time and date as prompted on the bottom line of the screen.

Note: The time and date is input without any separator characters. The Clock Driver is not updated unless the "TIME/DATE" key is pressed.

The Clock interrupt handler performs all functions necessary to maintain the system clock and all the calls necessary for peripheral timer routines. It then invokes User interrupt FF hex before returning.

The Configuration table does not hold any data related to the Clock driver.

Sound Driver 3.8

Overview

The Sound Generator hardware within the Apricot family is used by the Sound driver to produce the following:

Key click

Bell

Volume, Frequency and Period setting

Calls are provided through the Control Device for these facilities.

Applications Interest

In order to produce more complex Sound features such as music, the hardware must be accessed directly by the Application.

The Hardware section describes in detail how to access the Sound Generator directly.

Systems Interest

In conjunction with the Keyboard driver, the Sound driver produces a Key Click whenever a key is detected as having been depressed. This is important from the point of view of using infra-red keyboards. If the signal is not detected when a key is pressed, a Key Click is not generated.

It thus acts as an indication to the user of correct/incorrect positioning of the Keyboard relative to the Systems Unit.

Disk Driver 3.9

Contents

Overview

Applications interest

Non-MSDOS systems
Drive types
Label Sector
MS-DOS format
Disk format
Disk Swapping

Systems interest

Configuration data

Overview

The Disk Driver provides all functions for low level access of the Disks required by MS-DOS.

The Control Device calls to both Floppy and Winchester are designed to provide support at Operating System level only, i.e. for implementation of MS-DOS and other Operating Systems (OS).

Under normal circumstances all other Applications should be limited to use of the OS calls themselves and should NOT access the Control Device directly, unless otherwise stated in this chapter.

There are some exceptional cases. For example, the detection of "Disk swapped" is of importance to many Applications. Here the facilities are freely available within the Control Device. However they must be used with care. Refer to the section Disk swapping and to the Control Device chapter for the exact course of action to take.

The Disk Driver and Control Device support a number of different drive types and mixed combinations of Floppy and Winchester. Some drives however are hardware options. For full details, refer to the Hardware section.

The format of Disks is considered at two levels, i.e. Hardware physical format and Software MS-DOS format. Software "hooks" are provided within the BIOS to format a Floppy. Utilities are provided with the system software to perform this function for MS-DOS. Applications which require specific formatting should refer to the Hardware section.

The MS-DOS format is provided for completeness only. It depicts the actual layout of reserved areas on the disks, especially BOOT disks. The 'actual' format of any one Floppy or Winchester is defined in the Label Sector. Reference to the Guide to the BIOS chapter together with the appropriate MS-DOS manuals will provide full details.

Applications interest

Non MS-DOS systems

These applications are considered to be the implementation of other Operating Systems, for example CP/M.

The Control Device provides the facilities required to:

Detect the drive type

Format Floppy

Read/Write/Verify/Read + Verify sector(s)

Return disk status

Set/detect disk swapped

Return the BPB (BIOS Parameter Block) of a specified disk

The ROM Boot routine expects certain data within the Label Sector and minimum data in the BPB (refer to Label Sector section in this chapter and the Guide to the BIOS chapter).

Further items should be taken into account when preparing a Bootable disk:

The default printer device is selected from Label Sector offset 83 hex as the Parallel or Serial port.

The screen line (i.e. 200 or 256) and column (i.e. 40 or 80) mode is determined from the Label Sector offsets A0 and A1 hex respectively.

The Serial port is pre-defined in Label Sector offset B0 to CF hex.

Other sections of the Label Sector may be required to be preset but are not vital to the Boot procedure and will be OS dependant.

Drive types

The following drives are supported by the BIOS and Control Device:

Floppy : 70 Track SS
80 Track SS (not currently used)
80 Track DS

Winchester : 5 Megabyte
10 Megabyte
20 Megabyte (not currently used)

Up to 4 disks may be connected at any one time to the Apricot. Combinations of Floppy and Winchester are supported up to a maximum of 2 drives each.

Refer to the Hardware section for details of available options.

As mentioned in the MS-DOS section later the layout of a disk varies according to the Cluster (or allocation units) size. A Cluster is the smallest unit of the disk which is allocated to any file or reserved area on the disk.

The Cluster size is designed to use the disk as efficiently as possible and in general becomes larger as the capacity of the disk increases.

Label Sector

The Label Sector is the first physical sector on the Disk, i.e. Track 0 Sector 0. It has a length of 512 bytes.

The format of the complete sector is described in the chapter Guide to the BIOS.

The first 80H bytes of the Label Sector contain data relevant to OS Booting and Disk Configuration in the form of a BPB (BIOS Parameter Block). It provides the BIOS with a description of each disk (whether a Bootable Disk or not).

The BPB for each loaded disk is held in an array table in the ROM specified RAM. The Pointer area maintains a pointer to the Floppy and Winchester BPB array tables.

The BPB has the following format:

```
** WORD sector size in bytes
*  BYTE  sectors per allocation unit
   WORD number of reserved sectors
   BYTE  number of FAT's
   WORD number of directory entries
** WORD total number of sectors
   BYTE  Media descriptor byte: FCH = 70 SS
                                FDH = 80 SS
                                FEH = 80 DS
                                F8H = 5 megabyte
                                F9H = 10 megabyte
                                FAH = 20 megabyte
   WORD sectors per FAT
** BYTE  disk type:            0 = 70 track SS
                                1 = 80 track SS
                                2 = 80 track DS
                                3 = 5M winchester
                                4 = 10M winchester
                                5 = 20M winchester
   WORD Reserved
```

Notes:

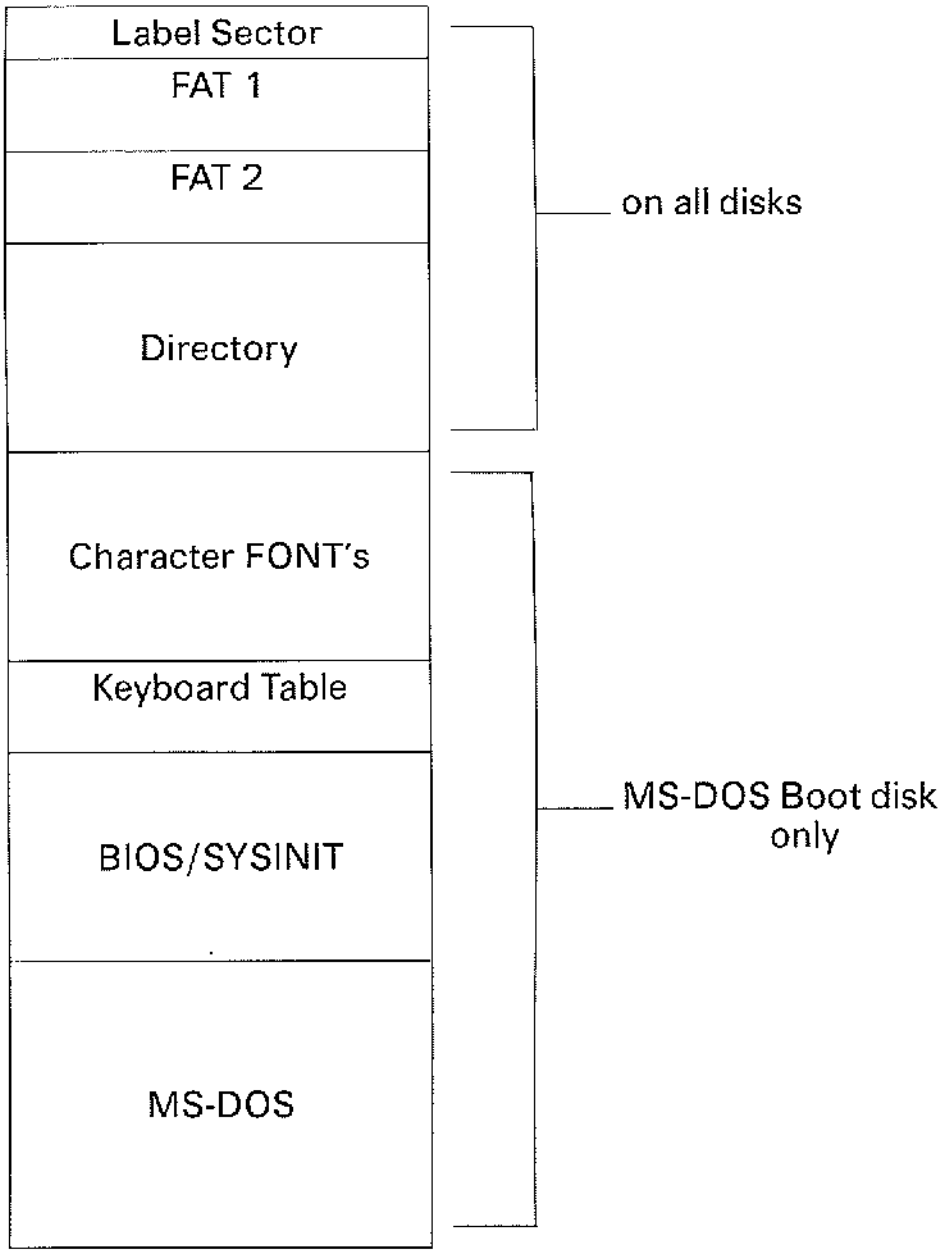
** These parameters are mandatory for the BOOT routines.

* This parameter must be non-zero.

All other parameters are OS specific, in this case MS-DOS.

MS-DOS format

The reserved part of each BOOT disk is as follows:



Only one 512 byte sector on the disk is specifically reserved and that is the first physical sector - the Label Sector.

All other information above is expected to be on the MS-DOS BOOT disks.

The location of each of the above is determined from data within the Label Sector itself.

The size of each section varies according to the Release of Software and to the cluster (or allocation) unit size of the disk.

MS-DOS 2.11 is about 17K whereas MS-DOS 3.0 is about 28K bytes. The cluster size for Winchester is bigger than for Floppy etc.

For example, the Keyboard Table is 1K on the Floppy Boot disks and 4K on the Winchester 5 megabyte. Only 1K is loaded into the Memory (and therefore the Map given in the Guide to the BIOS is always correct.)

Preparation of a Boot disk for other Operating Systems involves setting up the Label Sector exactly as required by the ROM BIOS Bootstrap routines.

Disk Formats

The Control Device provides facilities for formatting Floppy disks but there is no support for Winchester.

Formatting of a Winchester must be handled by Application or System Software alone. The BIOS does however reference a flag in the configuration table which enables or disables a Winchester park flag. (Refer to the Guide to the BIOS chapter 00 for details - Label sector offset EE hex).

The BIOS checks the Winchester for activity every 5 secs and if there has been no activity then the driver moves the heads to the last data track.

This is to protect the data tracks at the beginning of the disk if there is an inadvertant fault. This flag ensures that the BIOS does not park the heads during the format procedure.

For details of the actual physical format refer to the relevant Hardware section.

Disk Swapping

The Disk driver has a background "Demon" which monitors the condition of the drives every 2 seconds to see whether the disk is present.

The "Demon" assumes that a disk cannot be swapped faster than this.

A number of calls are provided in the Control Device to support disk swapping. If used, however, care must be taken to ensure that MS-DOS or applicable OS are aware of the present state.

Use of the Control Device call 0009 hex (check if disk in drive x swapped) results in the BIOS flag being reset. An application making this call should therefore restore the flag to its setting if necessary by immediately calling Call 0006H.

If MS-DOS detects a disk as swapped then all internal buffers etc related to it are flushed.

A further call is available in the Control Device to obtain the BPB of a disk in a specific drive.

Systems software should invoke this call following a disk swap to obtain correct disk information.

Configuration data

The configuration data for disks is located in two sections of the Label Sector.

The first 80 hex bytes of a Label Sector provide BOOT detail (where applicable) together with a BIOS Parameter Block (BPB). It is important to note that the BPB also contains information vital to the BOOT procedure - refer to Label Sector section.

Further data is to be found in the Label Sector, which is related to the Winchester only, the offsets and data are as follows:

Offset	Field	Bytes	Use
00E0		14	spare
00EE	CNF__wini__park	1	parking enable flag (0 = on, nz = off)
00EF	CNF__wini__form	1	format protection (0 = off, nz = on)
0100	WINbad__sect	64	Up to 32 words giving logical sector numbers of bad blocks on the disk
0140	WINvol__bpb1	16	Extended BPB for volume 1
0150	WINvol__bpb2	16	" " " vol 2
0160	WINvol__bpb3	16	" " " vol 3
0170	WINvol__bpb4	16	" " " vol 4
0180	WINvol__bpb5	16	" " " vol 5
0190	WINvol__bpb6	16	" " " vol 6
01A0	WINvol__bpb7	16	" " " vol 7
01B0	WINvol__bpb8	16	" " " vol 8

A non-dedicated area of label sector starting at offset 100H is used for Winchester disk bad block tables and reserved for Multi-volume BPB's.

Configuration data for each Floppy and Winchester known to the system is loaded into RAM and may be referenced by the Pointer table in 'ROM specified RAM', as follows:

Address	Length (bytes)	Set by	Use
408H	2	BOOT	Drive booted from: 0000H = Floppy 0 0001H = Floppy 1 0002H = Winchester 1 0003H = Winchester 2
40AH	4	BOOT	Pointer to loaded boot disk header sector
40EH	4	BOOT	Reserved
412H	2	BOOT	Reserved
414H	1	BOOT	Winchester type: 0 = No Winchester 3 = 5 Megabyte R0351 Winchester 4 = 10 Megabyte R0352 Winchester 5 = 20 Megabyte Winchester
415H	1	BOOT	Floppy type: 0 = 70 track SS 1 = 80 track SS 2 = 80 track DS
416H	1	BOOT	Number of Floppy Drives
417H	1	BOOT	Number of Winchester Drives
418H	4	BOOT	Pointer to Floppy BPB array Table
41CH	4	BOOT	Pointer to Winchester BPB array Table

GSX - GIOS Details 3.10

Contents

Overview

Applications interest
- Displays
- Input Devices
- Calling GSX
- Additions to GSX 1.3

Systems interest
- System files

Overview

All graphics devices are inherently different. Displays, plotters and printers are all capable of drawing lines, filling in areas and producing various forms of text. They often achieve each function in a unique way, which implies that the software controlling these devices is never compatible.

Graphics System Extension (GSX) is a support package which provides Applications with a standard interface to graphics devices.

The GSX package consists of two major components, they are:

GDOS - Graphics Device Operating System from Digital Research

GIOS - Graphics Input Output System

The GIOS is a device driver which is loaded by GSX to perform the unique features of the individual screens. A GIOS is required for each different screen resolution used in the system.

GSX loads one GIOS at a time, commencing with the default GIOS (refer to Systems interest). If the Application requests, another available device driver can be loaded instead.

GDOS and the Application communicate with a set of commands which relate to a Normalised coordinate space in the range 0 - 32767 on an X and Y axis. GDOS scales the coordinates according to information provided by the resident GIOS device driver when it is installed.

For details on how to install and use Graphics System Extension refer to Digital Research's GSX-86 Programmer's manual.

The Applications interest section of this chapter details how to call GSX functions. A number of additional functions have been included in the GIOS device drivers for the GSX-86 Release 1.3 on all Apricots. These implement, as closely as possible, new functions for GSX-86 Release 2.0 together with some ACT specific functions.

The Systems interest section details the various options implemented within the GIOS and how to install them.

Application interest

Display features

For a complete description of the Application interface to GSX, you should refer to the GSX-86 Programmers manual.

The implementation of the GIOS for the Apricot Portable and F1 Displays has the following features:

Line attributes: 7 line styles:

1 - Solid
2 - Long dash
3 - Dot
4 - Dash dot
5 - Dash
6 - Dash dot dot
7 - User defined

1 line width (1 Pixel)

Marker attributes: 5 markers (1 to 5, i.e . + * · o x)

Size: 1 - n (where n = vertical resolution i.e. 200/256 scan lines)

Text attributes: 2 character fonts:

BLOCK the current Master font as selected in the pointer table. The height is fixed according to the font, i.e. 8 or 10 pixels.

STROKE a line drawing font based upon coordinate points. The height is limited by the number of Display lines.

Rotation in multiples of 1 degree. (for STROKE only).

Fill attributes: 4 interior styles

0 - Hollow
1 - Solid
2 - Pattern
3 - Hatch

General Drawing Primitives: 4 types:

Arc
Pie Slice
Circle
Bar

All radius specifications assume an extent (distance) in the x-axis.

Colour: Portable: 640 x 200 LCD monochrome
640 x 200/256 4 colour
640 x 200/256 8 colour
640 x 200/256 16 colour

F1 : 640 x 200/256 4 colour
320 x 200/256 16 colour

The default colours are as follows:

0 Black
1 White
2 Red
3 Green
4 Blue
5 Cyan
6 Yellow
7 Magenta
8 Grey
9 Light Blue
10 Light Green
11 Light Cyan
12 Light Red
13 Light Magenta
14 Brown
15 Low-intensity White

Input devices

Input Locator: the input Locator is either:

1. Mouse, or
2. Keyboard cursor position keys

Input Valuator: the Valuator maintains a continuous count of the depression of the '+' and '—' keys. The increment is 1 unless a numeric key is pressed - in the event the increment becomes the value of the key pressed. Any non-numeric key will terminate input.

Input Choice: a value in the range 0 to 99 may be entered. If a single digit is entered then terminate input with any non-numeric key.

Input String: a string of characters terminated by carriage return.

Escapes: all Escape Op codes are implemented.

Cell arrays: Cell array display is implemented.

Cell array inquiry is NOT implemented.

Calling GSX

A detailed description of the commands, parameters and method of calling the GDOS is given in the GSX-86 Programmer's manual.

The actual Call procedure is summarised below for Application interest:

There is only one Call to the GDOS via interrupt E0 hex with the following register settings:

CX - 0473H Function code

DS:DX - Segment:Offset pointer to a parameter block which contains the following five double word Segment:Offset pointers:-

PB	control array
PB + 4	input parameter array
PB + 8	input points array
PB + 12	output parameter array
PB + 16	output points array

The next section describes the specific changes for features implemented to conform to GSX-86 Release 1.3 and some additional ACT specific features.

The notation used in the GSX-86 Programmers manual is adopted in the calling sequences outlined in the next section, i.e.

contrl(x) - the control array of x integer elements

intin(x) - the input parameter array

ptsin(x) - the input coordinate points array

intout(x) - the output parameter array

ptsout(x) - the output coordinate points array

Additions to GSX 1.3

A number of additional functions are included within the GSX Release 1.3 for all Apricots.

These are mainly to conform with GSX-86 Release 2.0 specification although a few are specifically ACT functions.

Note: These additions will not necessarily be supported by ACT in future releases and are included in this manual for completeness only.

1. Set user defined line style
2. Bit block move (ACT only)
3. Set fill interior style
4. Set fill perimeter visibility
5. Exchange fill pattern
6. Fill rectangle
7. Exchange mouse form
8. Show cursor
9. Hide cursor
10. Sample mouse button states
11. Prestel operations (ACT only)

These functions together with their specific calling procedures are detailed in the remainder of this section.

Set user defined line style

This function sets the current user-defined line style pattern word in the device driver to the value in the specified pattern word, 16 bits.

The most significant bit (MSB) of the pattern word is the left most pixel displayed. This line style is used for subsequent polyline operations when the application selects user-defined line style, index 7.

Input

Contrl(1) - Escape opcode = 5
Contrl(2) - Number of input vertices = 0
Contrl(6) - Opcode = 52

intin(1) - User line style

Output

Contrl(3) - 0

Bit Block Move (ACT specific routine)

A facility has been provided for moving bit aligned blocks of data between the display and memory, using the standard GSX interface.

The routine is an escape operator with function code of 51.

Input

contrl(1) - Escape opcode = 5
contrl(2) - Number of input vertices = 10
contrl(4) - 4
contrl(6) - Function code = 51
contrl(7) - Offset of dest. bitmap
contrl(8) - Seg. of dest. bitmap
contrl(9) - Offset of srce. bitmap
contrl(10) - Seg. of srce. bitmap
contrl(11) - Offset of pattern
contrl(12) - Seg. of pattern
contrl(13) - Raster operation

intin(1) - Dest. bitmap type
intin(2) - Srce. bitmap type

ptsin(1) - Dest. origin x
ptsin(2) - Dest. origin y
ptsin(3) - Srce. origin x
ptsin(4) - Srce. origin y
ptsin(5) - Extent in x direction
ptsin(6) - Extent in y direction
ptsin(7) - Dest. bitmap width in NDC units
ptsin(8) - Dest. bitmap height in NDC units
ptsin(9) - Srce. bitmap width in NDC units
ptsin(10) - Srce. bitmap height in NDC units

Output

contrl(3) - 0

intout(1) - 0 : No errors detected
1 : Illegal source description
2 : Illegal destination description
3 : Illegal rasterop requested
4 : Illegal extent in x direction
5 : Illegal extent in y direction

The following assumptions are made:

All clipping is done by the application.

Source and destination rectangles are the same size.

Patterns are specified as 8 adjacent bytes with bits organised with the most significant bit displayed first.

If an address parameter is not required nothing need be passed. An address need not be supplied for the display.

All parameters are left unchanged.

Note: The source and destination rectangles may overlap.

Set Fill Interior Style

This function sets the fill interior style used in subsequent polygon fill operations. If the application requests an unavailable style, the area is hollow filled. GSX returns the selected style to the application. Hollow style fills the interior with the current background colour, index 0. Solid style fills the area with the currently selected fill colour.

Input

contrl(0) Opcode = 23
contrl(1) Number of input vertices = 0
contrl(3) Length of intin array = 1
contrl(6) Device handle

intin(0) Request fill interior style

0 - Hollow
1 - Solid
2 - Pattern
3 - Hatch
4 - User defined style

Output

contrl(2) Number of output vertices = 0
contrl(4) Length of intout = 1

intout(0) Fill interior style selected

Set Fill Perimeter Visibility

This function turns on or off the outline of a fill area. When visible, (the default at Open Workstation for this function), the border of a fill area is drawn in the current fill area colour with a solid line. When invisible, no outline is drawn. Any nonzero value of the visibility flag causes the perimeter to be drawn.

Input

contrl(0) Opcode = 104
contrl(1) Number of input vertices = 0
contrl(3) length of intin array = 1
contrl(6) Device handle

intin(0) Visibilty flag

zero	invisible
nonzero	visible

Output

contrl(2) Number of output vertices = 0
contrl(4) Length of intout array = 1

intout(0) Visibility selected

Exchange Fill Pattern

This function redefines the user-definable fill pattern and returns the old user-definable fill pattern definition. The inputs to this routine are two long pointers. The first points to a Fill Pattern Definition Block (FPDB), which defines the new fill pattern. The second points to a FPDB that will be overwritten with the old fill pattern. If the second pointer is zero, the old FPDB is not returned.

The format of a FPDB is shown below.

Reserved for future use, must be 1
Reserved for future use, must be 1
Reserved for future use, must be 0
16 bits by 16 bits of pattern data

For the pattern data, bit 15 of word 1 is the upper left hand bit of the pattern. Bit 0 of word 16 is the lower right hand bit of the pattern. Bit 0 is the least significant bit of the word. A bit value of 1 indicates foreground colour and a bit value of 0 the background colour. The colour used for the foreground is determined by the current fill area colour index.

The defined pattern is referenced by the Set Fill Interior Style function as style 4 and in the Fill Rectangle function.

Input

contrl(0)	Opcode = 112
contrl(1)	Number of input vertices = 0
contrl(3)	Length of intin array = 0
contrl(6)	Device handle
contrl(7 - 8)	Double word address of the user defined fill pattern
contrl(9 - 10)	Double word address of the area to copy the old fill pattern

Output

contrl(2)	Number of output vertices = 0
contrl(4)	Length of intout array = 0

Fill Rectangle

This function fills a rectangular area with the pattern defined with the Set Fill Pattern function. The rectangle is filled using the writing mode specified in intin(0).

Input	
contrl(0)	Opcode = 114
contrl(1)	Number of input vertices = 2
contrl(3)	Length of intin array = 1
contrl(6)	Device handle
contrl(7-8)	Double word address of the destinations memory form definition block (Not implemented, physical display assumed)
intin(0)	Writing mode
ptsin(0)	X-coordinate of upper left of destination rectangle in RC/NDC
ptsin(1)	Y-coordinate of upper left of destination rectangle in RC/NDC
ptsin(2)	X-coordinate of lower right of destination rectangle in RC/NDC
ptsin(3)	Y-coordinate of lower right of destination rectangle in RC/NDC
Output	
contrl(2)	Number of output vertices = 0
contrl(4)	Length of intout array = 0

Exchange Mouse Form

This function redefines the cursor pattern displayed during locator input or any time the cursor is shown (see Show Cursor function), and returns the old cursor definition. The inputs to this routine are two long paramaters. The first points to a cursor definition block which is to be used as the new cursor. The second points to a cursor definition block which will be overwritten with the old cursor definition block. If the second pointer is zero, the old cursor definition block is not returned.

Format of Cursor Definition Block

X-coordinate of hot spot
Y-coordinate of hot spot
Reserved, must be 1
Colour Index
Reserved, must be 0
16 by 16 bit of cursor mask
16 by 16 bit of cursor data

For the cursor mask and data, bit 15 of word 1 is the upper left hand bit of the pattern and bit 0 of word 16 is the lower right bit of the pattern. Bit 0 is the least significant bit (LSB) of the word. A bit value of 1 indicates foreground colour and a bit value of 0 indicates background colour. The colour used for foreground colour is determined by the colour index.

The hotspot is the location of the pixel (relative to the upper left corner of the cursor) that lies over the pixel whose address is returned by the input locator function.

The cursor is drawn as follows:

The data under the cursor is saved so it can be restored when the cursor moves.

The mask is ANDed with the data on the screen.

The cursor data is XORed with the result of the previous step and displayed on the screen.

Input	
contrl(0)	Opcode = 111
contrl(1)	Number of input vertices = 0
contrl(3)	Length of intin array = 0
contrl(6)	Device handle
contrl(7 - 8)	Double word address of the new cursor definition block
contrl(9 - 10)	Double word address to receive the old cursor definition block
Output	
contrl(2)	Number of output vertices = 0
contrl(4)	Length of intout array = 0

Show Cursor

This function displays the current cursor. The cursor moves on the display surface based on information input from a mouse or the cursor control keys.

The Show Cursor function and the Hide Cursor function are closely related. Once the cursor is visible, a single Hide Cursor causes the cursor to disappear. GSX maintains the number of times the Hide Cursor function is called. The Show Cursor function must be called the same number of times for the cursor to reappear. For example, if the Hide Cursor function is called four times, the Show Cursor function must be called four times for the cursor to reappear.

However, the Show Cursor function provides a reset flag in intin(0). If intin(0) is zero, the cursor appears on the screen reguardless of the number of Hide Cursor calls. A nonzero value for intin(0) affects the Show Cursor function as described in the previous paragraph.

Input

contrl(0) Opcode = 122
contrl(1) Number of input vertices = 0
contrl(3) Length of intin array = 1
contrl(6) Device handle

intin(0) Reset flag

0 = ignore number of hide cursor calls
nonzero = normal show cursor behavior

Output

contrl(2) Number of output vertices = 0
contrl(4) Length of intout array

Hide Cursor

This function removes the cursor from the display surface. This state is the default condition set at Open Workstation. The cursor can appear in a new position when the application calls the Show Cursor function because GSX updates the position based on information input from a mouse or the cursor control keys.

Refer to the Show Cursor function for a description of how the number of Show Cursor calls affects the Hide Cursor function.

Input

contrl(0) Opcode = 123
contrl(1) Number of input vertices = 0
contrl(3) Length of intin array = 0
contrl(6) Device handle

Output

contrl(2) Number of output vertices = 0
contrl(4) Length of intout array = 0

Sample Mouse Button State

This function returns the current state of the mouse buttons. The leftmost mouse button is returned in the least significant bit of the word. A bit value of 1 indicates the key is currently depressed, a bit value of 0 indicates the key is up.

This function also returns the current x, y position of the cursor.

Input

contrl(0) Opcode = 124
contrl(1) Number of input vertices = 0
contrl(3) Length of intin array = 0
contrl(6) Device handle

Output

contrl(2) Number of output vertices = 1
contrl(4) Length of intout array = 1

intout(0) Mouse button state

ptsout(0) X position of cursor in NDC/RC units
ptsout(1) Y position of cursor in NDC/RC units

Prestel Operations. (ACT specific routine)

An escape op is provided to allow the setting of attributes for PRESTEL text. The only attribute initially available is double height. Colour is handled using standard GSX text functions.

Input

Contrl(1) Escape opcode = 5
Contrl(2) Number of input vertices = 0
Contrl(6) Op code = 55

intin(1) Set text single height = 0
Set text double height = 1

Output

Contrl(3) 0

An escape operation is also provided to allow the writing of PRESTEL text. This operation depends on the standard font pointer refering to a double width font for 40 column mode PRESTEL.

Input	
contrl(1)	Escape opcode = 5
contrl(2)	Number of input vertices = 1
contrl(4)	Number of characters in string
contrl(6)	Opcode = 56
intin	Word character string = 0
ptsin(1)	X-coordinate for start of string
ptsin(2)	Y-coordinate for start of string
Output	
contrl(3)	0

Notes:
This routine assumes that characters require no clipping. The x,y position for the character is the bottom left corner of the character cell NOT the character origin.

The writing mode is always transparent (mode 2) so that PRESTEL can use bar for the background and then draw 'OR' the character onto this.

System Files

The GSX system comprises 2 fixed files

GRAPHICS.EXE
ASSIGN.SYS

together with a variable number of GIOS device driver files which are defined in the ASSIGN.SYS file in a pre-determined format as discussed in the GSX-86 Programming manual.

The contents of ASSIGN.SYS may be simply inspected by invoking an MS-DOS command as follows:

```
A>TYPE ASSIGN.SYS

06 F116        .SYS  F1 16 colour 40 column
01 PC          .SYS  Apricot PC monochrome
02 F1          .SYS  F1 4 colour 80 column
03 PORTLCD     .SYS  Portable LCD
04 PORTCOL3    .SYS  Portable 8 colour
05 PORTCOL4    .SYS  Portable 16 colour

A>
```

In order to invoke a different default device edit the ASSIGN.SYS file as described in the GSX-86 Programmer's manual. Remember that ASSIGN.SYS may contain any number of GIOS Device drivers.

The first occurrence in the file is the Default GIOS driver and it should always be the largest of all required device drivers. It is loaded by GSX which only reserves sufficient memory for the default driver.

Applications may invoke other GIOS device drivers by using a standard GSX Call 'Open workstation' which references the file by the unique number which should be assigned to it in ASSIGN.SYS. Only one GIOS may be resident at a time.

Appendices

Appendix A - Diagnostics Error Codes

When the system is booted up a check is made to see whether it is a cold or warm start. If it is a cold (power-up) start the ROM resident diagnostic program is run to check the integrity of the hardware.

If an error is found the diagnostic program is terminated and the error number is shown on the start-up screen. See Table 1.

The ROM BIOS enters the boot procedure, displays the start-up screen and eventally starts looking for a bootable disk. Boot disk errors are listed in Table 2. Note that the screen display of an error from the diagnostics is overwritten by the boot procedure.

Both types of error are indicated by a large 'X' on the left of the screen, and a one or two digit error code on the right.

Table 1. Boot ROM test error codes

Code	Test Failed
20	ROM TEST: Calculates the checksum of the ROM and compares it to that stored at the start of ROM.
22	SIO TEST: Register test of Z80 SIO
25	SYSTEM RAM: Checks the system and any extension RAM.
28	FDC TEST: Register test of the WD 2797 floppy disk controller.
29	CTC TEST: Register test of the clock chip.
33	CLOCK INT: Checks whether a clock interrupt can be generated
35	DRIVE TEST: Checks whether drive 0 exists and can be restored and stepped.

Table 2. Disk error codes

Code	Reason
2	Drive not ready or disk removed during boot.
4	CRC error, corrupt sector format.
6	Seek error; unformatted or corrupt disk, or faulty floppy drive.
7	Bad media, corrupt media block.
8	Sector not found; unformatted or corrupt disk, or bad load address in label.
11	Bad read, corrupt data field on disk.
12	Disk failure, disk hardware or media fault.
99	Non-System disk (Not a valid boot disk, or disk incompatible with drive, or invalid load address). Note: if another loadable disk is found, this message will only be displayed very briefly.

Appendix B - Default Keyboard Table

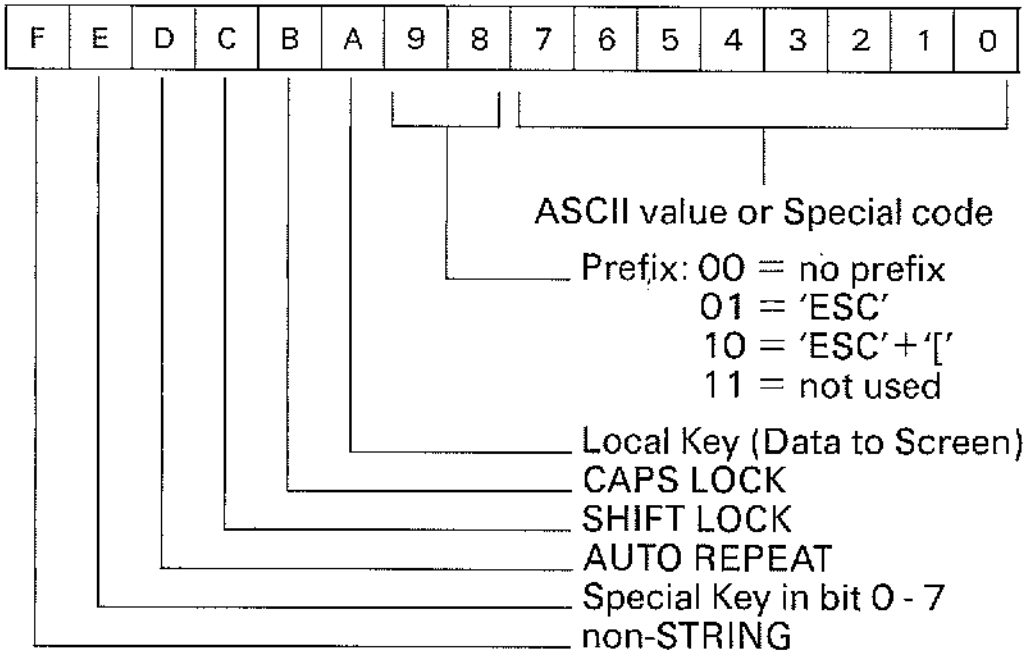
The Keyboard Table and its function is described in detail in the Keyboard Driver chapter. The purpose of this appendix is to provide a quick reference guide to the format of the ROM based keyboard table.

The default table is split into three sections each 104 words long to accomodate NORMAL, SHIFTed and CONTROL modes.

Although these 3 sections respectively follow each other in the ROM table, they are depicted below in "down-code" sequence as returned by the Keyboard Driver showing each of the three keyboard modes together.

Each entry in the table is 1 word in the following format:

(0) <— 15 bit STRING pointer —>


F E D C B A 9 8 7 6 5 4 3 2 1 0
ASCII value or Special code
Prefix: 00 = no prefix
01 = 'ESC'
10 = 'ESC'+'['
11 = not used
Local Key (Data to Screen)
CAPS LOCK
SHIFT LOCK
AUTO REPEAT
Special Key in bit 0 - 7
non-STRING

Attribute bits A - F are: 'ON' if set to 1
'OFF' if set to 0.

The exception to this format is when the "Non-STRING" attribute F is set 'OFF' implying that the down-code is to be translated into a string of data. In this case the format is:

bit 0 - bit E : String offset of 15 bits (i.e. 32K)
bit F : set 0 = STRING

The 15 bit address points to an entry which must be after the end of the first 312 words of the table.

The actual entry is of a different format as it is of variable length. The format is:

Byte 'n' : String byte count 'n'

Byte 'n' : Attributes

'n' Bytes : Data

Examples of String entries are to be found in the table in the format of a pointer as follows:

<string name>

e.g. <80 Column Apricot>

Note: The 6 Membrane Keys MB1 - MB6 are not available on the Portable or the F1.

		Normal		Shift		Control	
Legend	**Down code**	**Data**	**Attribs. Normal**	**Data**	**Attribs. Shift**	**Data**	**Attribs. Control**
F1/HELP	01	B1H	10000000	B2H	10000000	<80 col Apricot Comp.> (see Note 1)	
F2/UNDO	02	B2H	10000000	B2H	10000000	<80 col Colour>	
F3/REPEAT	03	B3H	10000000	'U'	10000001	<40 col Colour> (see Note 2)	
F4/CALC	04	B4H	10000000	06H	11000000	<80 col Apricot Comp.> (see Note 3)	
F6/PRINT	05	B5H	10000000	26H	10000101	07H	11000000
F7/INTR	06	B6H	10000000	B6H	10000000	B6H	10000000
F8/MENU	07	B7H	10000000	B7H	10000000	B7H	10000000
F9/FINISH	08	B8H	10000000	B8H	10000000	B8H	10000000
MB1	09	B9H	10100000	B9H	10100000	B9H	10100000
MB2	10	BAH	10100000	BAH	10100000	BAH	10100000
MB3	11	BBH	10100000	BBH	10100000	BBH	10100000
MB4	12	BCH	10100000	BCH	10100000	BCH	10100000
MB5	13	BDH	10100000	BDH	10100000	BDH	10100000
MB6	14	BEH	10100000	BEH	10100000	BEH	10100000
^	15	'^'	10110000	'^'	10110000	'^'	10110000
1!	16	'1'	10110000	'!'	10110000	ACH	10000000
2@	17	'2'	10110000	'@'	10110000	ABH	10000000
3#	18	'3'	10110000	'#'	10110000	00H	10000000
4#	19	'4'	10110000	'#'	10110000	9DH	10000000
5%	20	'5'	10110000	'%'	10110000	F8H	10000000
6$	21	'6'	10110000	'$'	10110000	9BH	10000000
7&	22	'7'	10110000	'&'	10110000	'''	10000000
8*	23	'8'	10110000	'*'	10110000	F1H	10000000
9(	24	'9'	10110000	'('	10110000	'~'	10110000
0)	25	'0'	10110000	')'	10110000	'\'	10110000
—_	26	'—'	10110000	'_'	10110000	'—'	10110000
=+	27	'='	10110000	'+'	10110000	'='	10110000
Backspace	28	08H	10100000	08H	10100000	08H	10100000
Percentage	29	'%'	10100000	'%'	10100000	'%'	10100000
Multiply	30	'*'	10100000	'*'	10100000	'*'	10100000
Divide	31	'/'	10100000	'/'	10100000	'/'	10100000
Subtract	32	'—'	10100000	'—'	10100000	'—'	10100000
Addition	33	'+'	10100000	'+'	10100000	'+'	10100000
TAB/BACK	34	09H	10100000	09H	10100000	09H	10100000

Notes:

1. This directs screen output to the default screen using Apricot Compatible mode. The default screen is the LCD on the Apricot Portable; the normal display monitor on the Apricot F1.
2. 40 column mode is supplied on the F1 only.
3. 80 column Apricot Compatible mode in any two colours. This is the same mode as defined by CTRL + F1 on the Apricot F1, but directs screen output to the colour monitor on the Portable (instead of the LCD).

		Normal		Shift		Control	
Legend	**Down code**	**Data**	**Attribs. Normal**	**Data**	**Attribs. Shift**	**Data**	**Attribs. Control**
Q	35	'q'	10111000	'Q'	10111000	11H	10111000
W	36	'w'	10111000	'W'	10111000	17H	10111000
E	37	'e'	10111000	'E'	10111000	05H	10111000
R	38	'r'	10111000	'R'	10111000	12H	10111000
T	39	't'	10111000	'T'	10111000	14H	10111000
Y	40	'y'	10111000	'Y'	10111000	19H	10111000
U	41	'u'	10111000	'U'	10111000	15H	10111000
I	42	'i'	10111000	'I'	10111000	09H	10111000
O	43	'o'	10111000	'O'	10111000	0FH	10111000
P	44	'p'	10111000	'P'	10111000	10H	10111000
[<	45	'['	10110000	'<'	10110000	'['	10110000
] >	46	']'	10110000	'>'	10110000	']'	10110000
HOME/ESC H	47	'H'	10000001	'H'	10000001	'H'	10000101
CLEAR	48	'E'	10000001	'E'	10000001	'z'	10000001
7	49	'7'	10111000	'7'	10111000	'7'	10100000
8	50	'8'	10111000	'8'	10111000	'8'	10100000
9	51	'9'	10111000	'9'	10111000	'9'	10100000
CAPS LOCK	52	01H	11000000	01H	11000000	01H	11000000
A	53	'a'	10111000	'A'	10111000	01H	10111000
S	54	'b'	10111000	'S'	10111000	13H	10111000
C	55	'd'	10111000	'D'	10111000	04H	10111000
F	56	'f'	10111000	'F'	10111000	06H	10111000
G	57	'g'	10111000	'G'	10111000	07H	10111000
H	58	'h'	10111000	'H'	10111000	08H	10111000
J	59	'j'	10111000	'J'	10111000	0AH	10111000
K	60	'k'	10111000	'K'	10111000	0BH	10111000
L	61	'l'	10111000	'L'	10111000	0CH	10111000
;:	62	';'	10110000	':'	10110000	';'	10110000
'"	63	'''	10110000	'"'	10110000	'''	10110000
RETURN	64	0DH	10100000	0DH	10100000	0DH	10100000
Line Insrt Char	65	40H	10100001	40H	10100001	40H	10100001
Line Del Char	66	7FH	10100000	7FH	10100000	7FH	10100000
4	67	'4'	10111000	'4'	10111000	'4'	10111000
5	68	'5'	10111000	'5'	10111000	'5'	10111000
6	69	'6'	10111000	'6'	10111000	'6'	10111000

		Normal		Shift		Control	
Legend	**Down code**	**Data**	**Attribs. Normal**	**Data**	**Attribs. Shift**	**Data**	**Attribs Control**
LH shift	70	03H	11000000	03H	11111000	03H	11000000
Z	71	'z'	10111000	'Z'	10111000	1AH	10111000
X	72	'x'	10111000	'X'	10111000	18H	10111000
C	73	'c'	10111000	'C'	10111000	03H	10111000
V	74	'v'	10111000	'V'	10111000	16H	10111000
B	75	'b'	10111000	'B'	10111000	02H	10111000
N	76	'n'	10111000	'N'	10111000	0EH	10111000
M	77	'm'	10111000	'M'	10111000	0DH	10111000
, <	78	','	10110000	'<'	10110000	','	10110000
. >	79	'.'	10110000	'>'	10110000	'.'	10110000
/ ?	80	'/'	10110000	'?'	10111000	'/'	10110000
RH shift	81	02H	11000000	02H	11000000	02H	11000000
Up arrow	82	'A'	10100001	<Fast screen scroll>		<LCD contrast up>	
Scroll	83	0AH	10100000	00H	10000000	00H	10000000
1	84	'1'	10111000	'1'	10111000	'1'	10111000
2	85	'2'	10111000	'2'	10111000	'2'	10111000
3	86	'3'	10111000	'3'	10111000	'3'	10111000
ESC	87	1BH	10100000	1BH	10100000	1BH	10100000
CONTROL	88	04H	11000000	04H	11000000	04H	11111000
SPACE	89	' '	10100000	' '	10100000	' '	10100000
STOP	90	05H	11000000	05H	11000000	05H	11000000
Left arrow	91	'D'	10100001	'D'	10100001	<Bell volume dn>	
Down arrow	92	'B'	10100001	<Smooth scrolling>		<LCD contrast down>*	
R arrow	93	'C'	10100001	'C'	10100001	<Bell volume up>	
0	94	'0'	10100000	'0'	10100000	'0'	10100000
.	95	'.'	10100000	'.'	10100000	'.'	10100000
ENTER	96	0DH	10100000	0DH	10100000	0DH	10100000
F5/VOICE	97	B9H	10000000	08H	11000000	08H	11000000
spare	98		11111000		11111000		11111000
spare	99		11111000		11111000		11111000
spare	100		11111000		11111000		11111000
spare	101		11111000		11111000		11111000
spare	102		11111000		11111000		11111000
spare	103		11111000		11111000		11111000
spare	104		11111000		11111000		11111000

* Apricot Portable only

String name	String Length	Attribs.	STRING		
<LCD contrast up>	3	10100100	27	'y'	'E'
<LCD contrast dn>	3	10100100	27	'x'	'E'
<Fast screen scroll>	3	10100100	27	'y'	'D'
<Smooth scrolling>	3	10100100	27	'x'	'D'
<Bell volume up>	3	10100100	27	'y'	'F'
<Bell volume dn>	3	10100100	27	'x'	'F'
<80 column Apricot>	3	10100100	27	'7'	'L'
<80 col Colour>	3	10100100	27	'7'	'C'
<40 col Colour>	3	10100100	27	'7'	'F'
<80 column Apricot>	3	10100100	27	'7'	'B'

The remainder of the 1024 bytes of the default table are zero.

Appendix C Ascii codes

The Apricot supports a default 128 character Ascii set in ROM and optionally a full 256 character Ascii set in RAM.

The full character set is tabled below in Figure 1.

Following power up the pre-Boot software uses the 128 character Font in the ROM.

The Bootstrap procedure searches for the file FONT.SYS which contains an 8 x 8 and an 8 x 10 Font to provide support for 200 and 256 scan lines respectively. If found then the file is loaded into RAM at a fixed position (Refer to Guide to the BIOS Memory Map).

A constant at offset 0A0H in the Disk Label Sector (refer to the chapter Guide to the BIOS) indicates the number of scan lines and hence the Font to be used. The Bootstrap procedure references this constant and then sets up pointers to the Master and Active Font.

The pointer table in the 'ROM specified RAM' (refer to Memory Map in chapter Guide to the BIOS) has two entries which indicate where the Master and Active Fonts are located. Each entry consists of the following:

Offset/Segment pointer to the character set

Length of the Font in bytes

The Master Font and Active Font pointers are set by the Bootstrap routine.

The Master Font pointer points to the base of the table in RAM where FONT.SYS is loaded.

If FONT.SYS is not on the disk then the Master Font and Active Font pointers both point to the ROM Font.

If FONT.SYS is on the disk, it is loaded into RAM and the Active Font pointer is set according to the parameter in the configuration data of the Label Sector. It will point to either the 8 x 8 or the 8 x 10 Font.

Normally Applications will not need to alter the pointers. However, Applications may use alternative/multiple character sets by loading them in free memory and modifying the Active pointer and length.

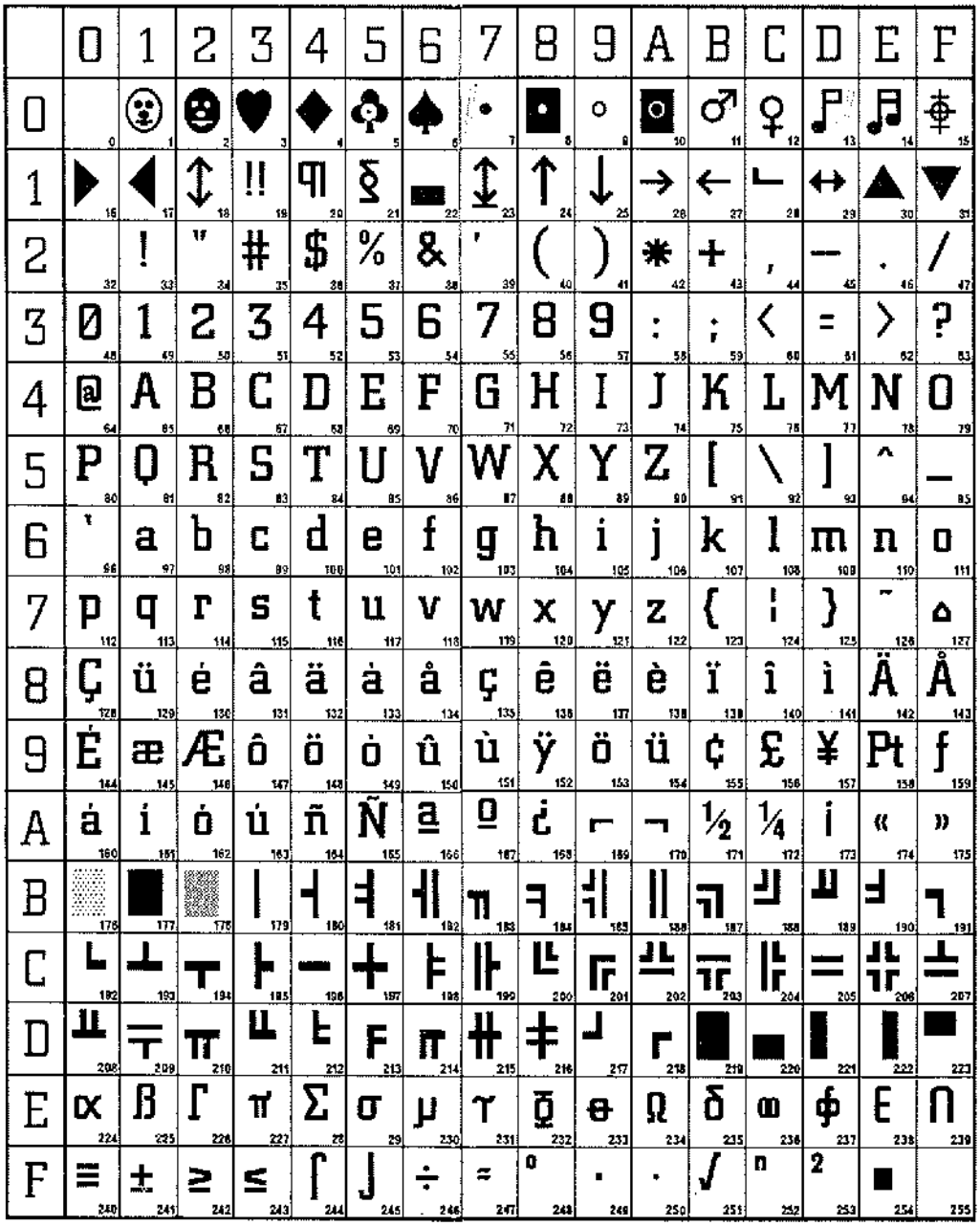


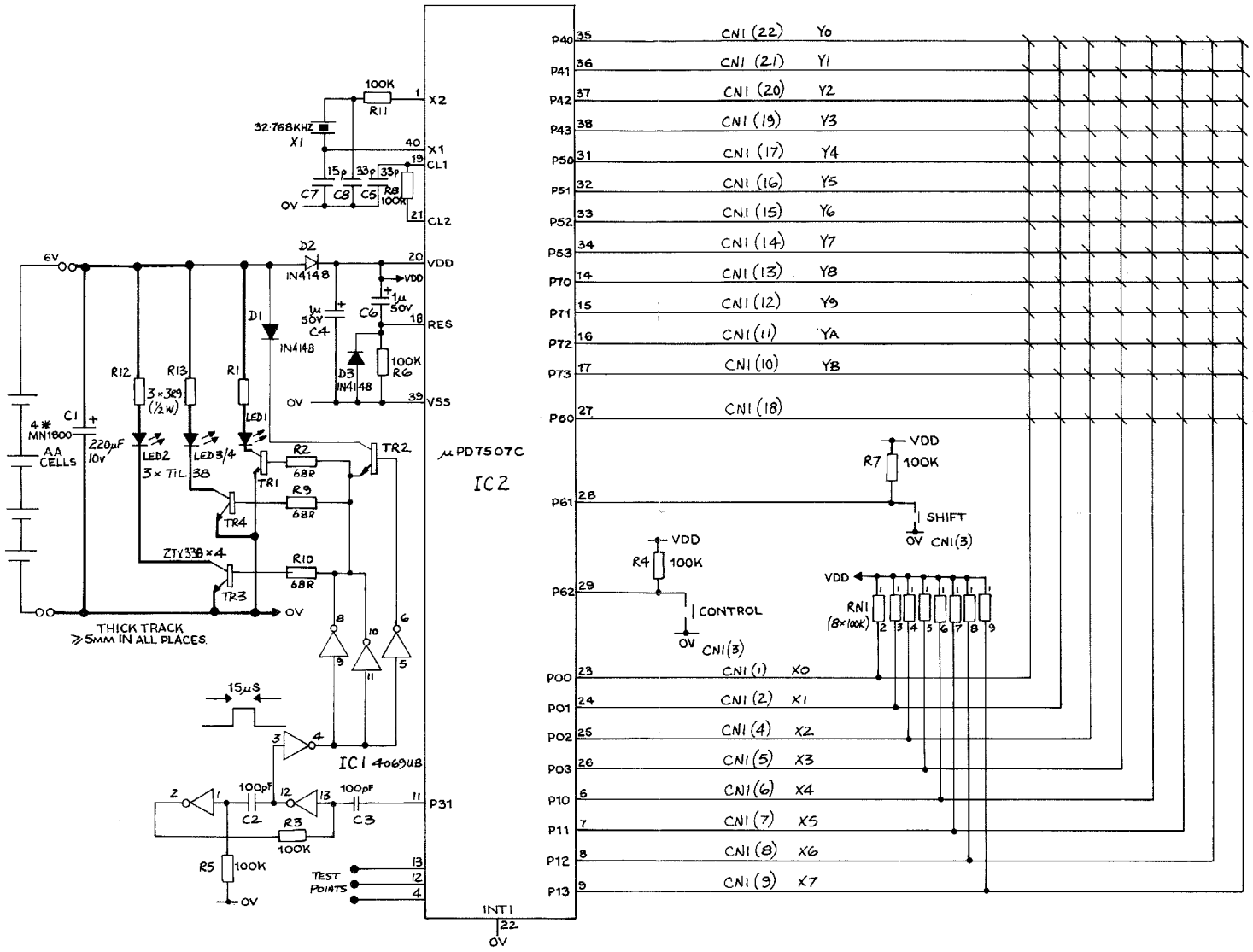
Figure 1. Default 256 character ASCII set

Appendix D — Circuit Diagrams

Figure 1. Apricot F1 — CPU and Peripherals
Figure 2. Apricot F1 — Memory and Display
Figure 3. Apricot F1 — 128K RAM section
Figure 4. Apricot F1 — Expansion Port
Figure 5. IR Receiver Board
Figure 6. Keyboard

3rd ANGLE PROJECTION

(CN1 ; 23 WAY SIL CONNECTOR)



CN057	C	21.6.84
CN042	B	14.5.84
CN037	A	12.4.84
ALT No	ISSUE	DATE
APPROVED		
CERTIFIED		
FIRST JOB No		

ACT (advanced technology) LTD
BIRMINGHAM.

This drawing and the design hereon are the exclusive property of A.C.T. The drawing shall not be copied nor its contents communicated or reproduced in any form and for any purpose without prior specific authority in writing of A.C.T.

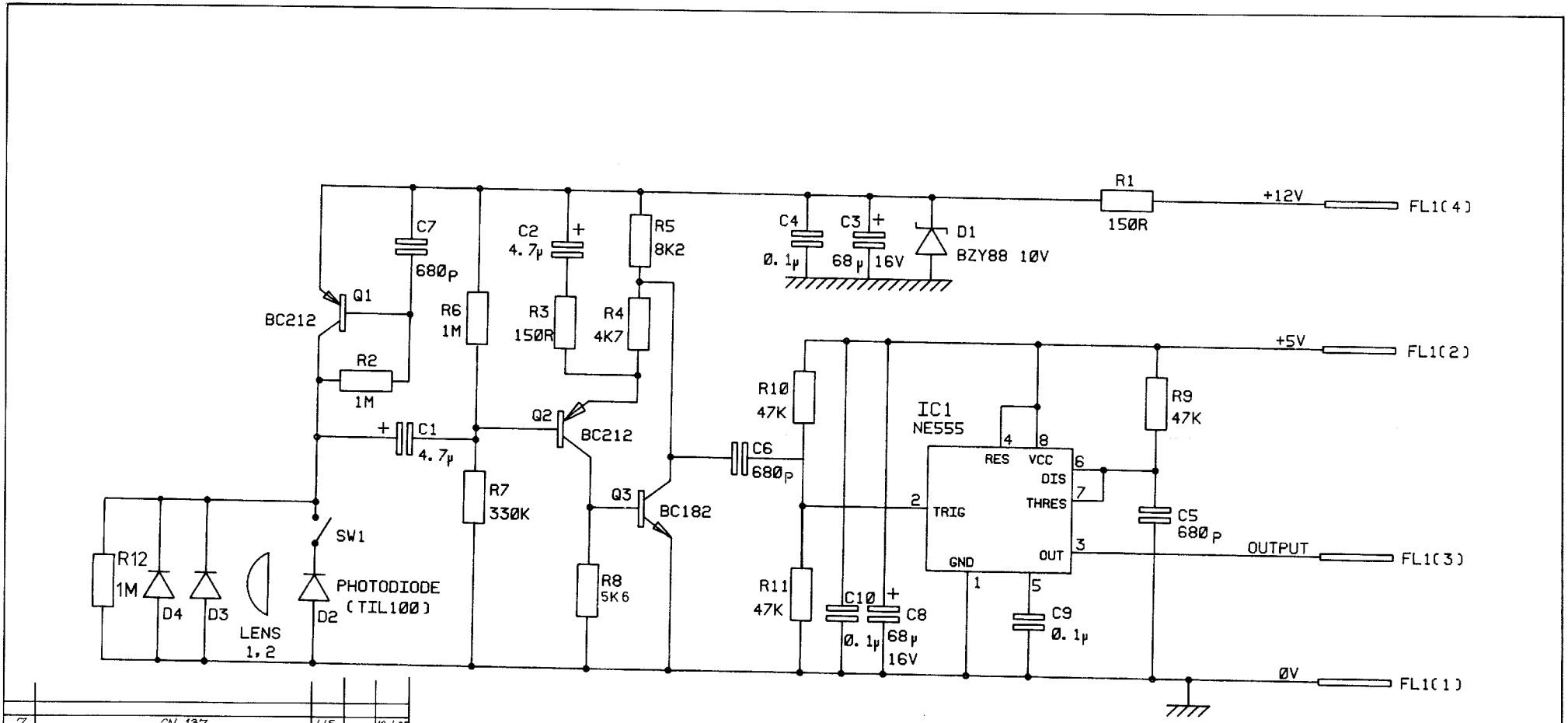
THIRD

SCALE	
DRAWN	RD
CHECKED	RD
DATE	22/2/85

TITLE: IR KEYBOARD

DRG. NO.

SHT 1 OF SHTS 1



7	CN. 137	JJS		10-1-85
6	REDRAWN	TB		12-7-84
5	CHANGE NOTE No. 065			28-6-84
4	CHANGE NOTE No. 051			13-6-84
3	CHANGE NOTE No. 036			12-4-84
2	CHANGE NOTE No. 035			11-4-84
1				17.11.83
No.	DESCRIPTION	CHK'D	APP'D	DATE

REVISIONS

ACT (advanced technology) LTD
BIRMINGHAM.

This drawing and the design hereon are the exclusive property of A.C.T. The drawing shall not be copied nor its contents communicated or reproduced in any form and for any purpose without prior specific authority in writing of A.C.T.

THIRD

SCALE

DRAWN RD

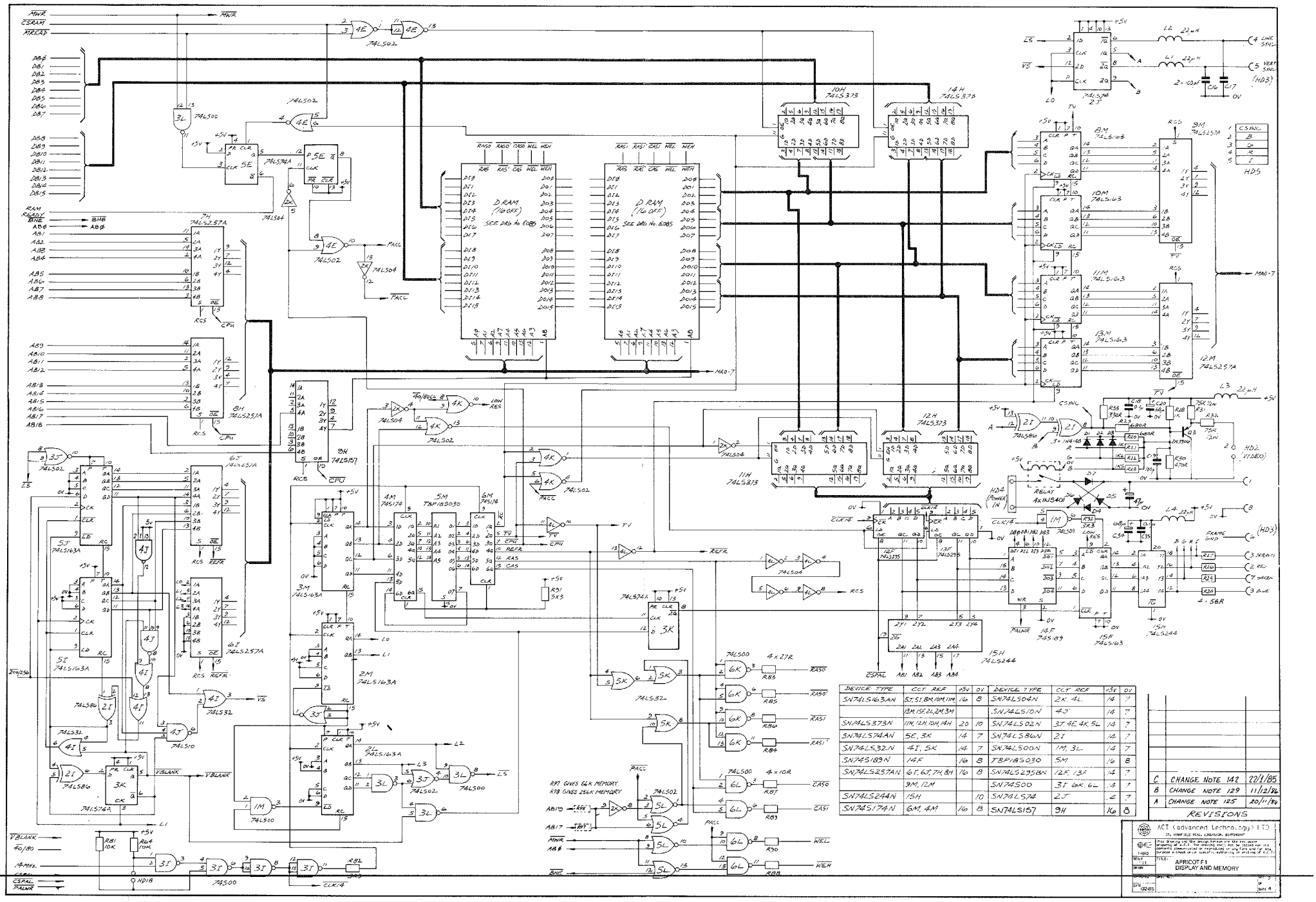
CHECKED RD

DATE 22/2/85

TITLE: INFRA RED DETECTOR

DRG. NO.

SHT 1 OF SHTS 1



DEVICE TYPE	CCT REF	+5V	0V	DEVICE TYPE	CCT REF	+5V	0V
SN74LS163AN	5T,5I,8M,10M,11M	16	8	SN74LS04N	2K 4L	14	7
	13M,15F,2L,2M,3M			SN74LS10N	4J	14	7
SN74LS373N	11M,12H 10H,14H	20	10	SN74LS02N	3J 4E 4K 5L	14	7
SN74LS74AN	5E, 3K	14	7	SN74LS86N	2I	14	7
SN74LS32N	4I, 5K	14	7	SN74LS00N	1M, 3L	14	7
SN74S189N	14F	16	8	TBP18S030	5M	16	8
SN74LS257AN	6I, 6J, 7H, 8H	16	8	SN74LS295N	12F, 13F	14	7
	9M, 12M			SN74S00	3I 6K, 6L	14	7
SN74LS244N	15H		10	SN74LS74	2J	14	7
SN74S174N	6M, 4M	16	8	SN74LS157	9H	16	8

C	CHANGE NOTE 142	22/1/85
B	CHANGE NOTE 129	11/12/84
A	CHANGE NOTE 125	20/11/84

REVISIONS

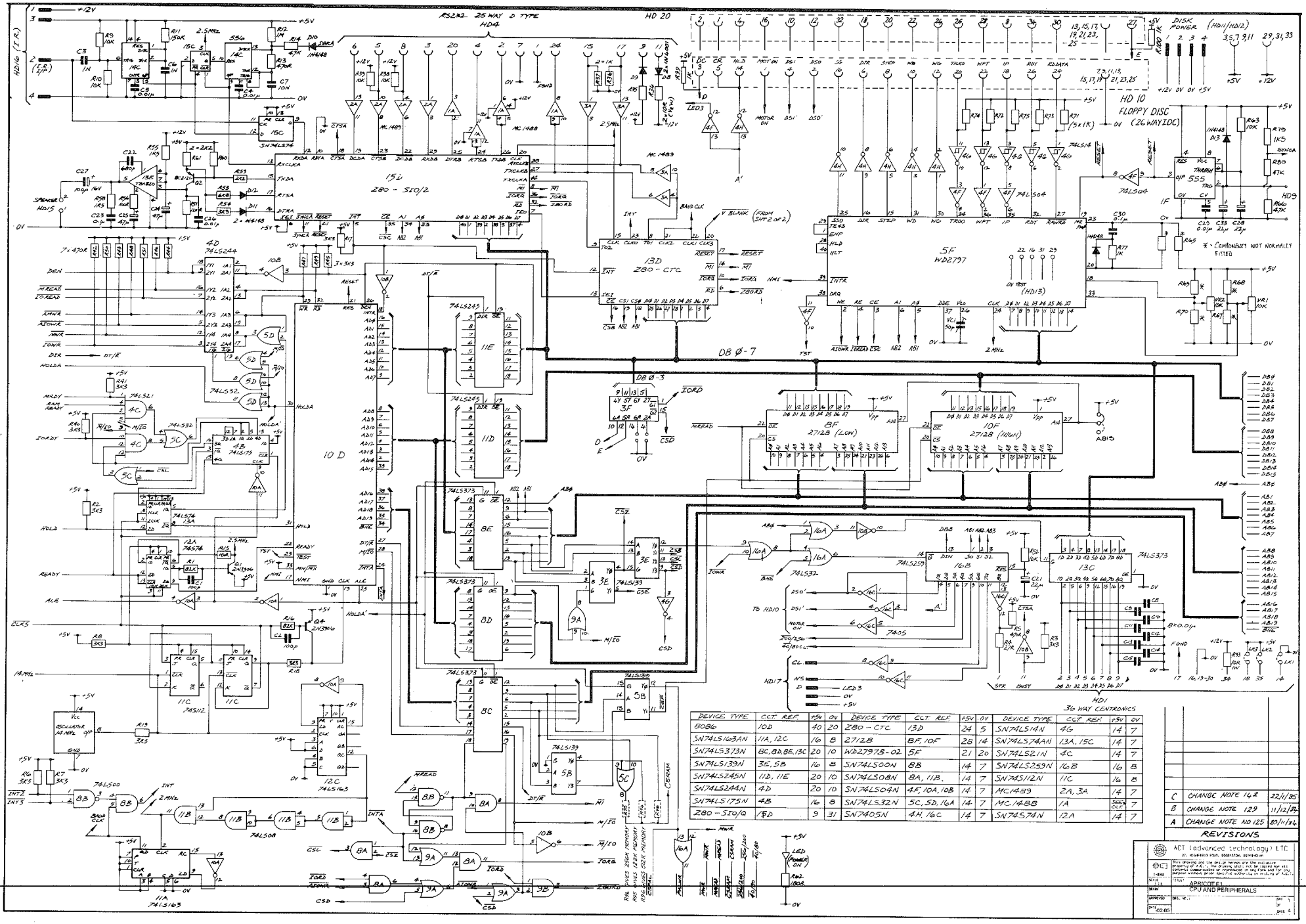
ACT Advanced Technology LTD
35, HEMPSTEAD ROAD, KINGS LANGLEY, HERTFORDSHIRE

This drawing and the design therein are the exclusive property of A.C.T. The drawing shall not be copied nor its contents communicated or reproduced in any form and/or used, purchased or sold without the written authority of A.C.T.

SIZE A3
DRAWN
DATE 02-85

TITLE: APRICOT F1
DISPLAY AND MEMORY

DRAWING No.
Sheet 4



DEVICE TYPE	CCT REF.	+5V	0V	DEVICE TYPE	CCT REF.	+5V	0V	DEVICE TYPE	CCT REF.	+5V	0V
8086	10D	40	20	Z80 - CTC	13D	24	5	SN74LS14N	4G	14	7
SN74LS103AN	11A, 12C	16	8	2712B	8F, 10F	28	14	SN74LS74AN	13A, 15C	14	7
SN74LS373N	8C, 8D, 8E, 13C	20	10	WD2797B-02	5F	21	20	SN74LS21N	4C	14	7
SN74LS139N	3E, 5B	16	8	SN74LS00N	8B	14	7	SN74LS259N	16B	16	8
SN74LS245N	11D, 11E	20	10	SN74LS08N	8A, 11B,	14	7	SN745112N	11C	16	8
SN74LS244N	4D	20	10	SN74LS04N	4F, 10A, 10B	14	7	MC1489	2A, 3A	14	7
SN74LS175N	4B	16	8	SN74LS32N	5C, 5D, 16A	14	7	MC1488	1A	14	7
Z80 - SIO/0	15D	9	31	SN7405N	4H, 16C	14	7	SN74S74N	12A	14	7

C	CHANGE NOTE 142	22/1/85
B	CHANGE NOTE 129	11/12/84
A	CHANGE NOTE NO 125	20/11/84

REVISIONS

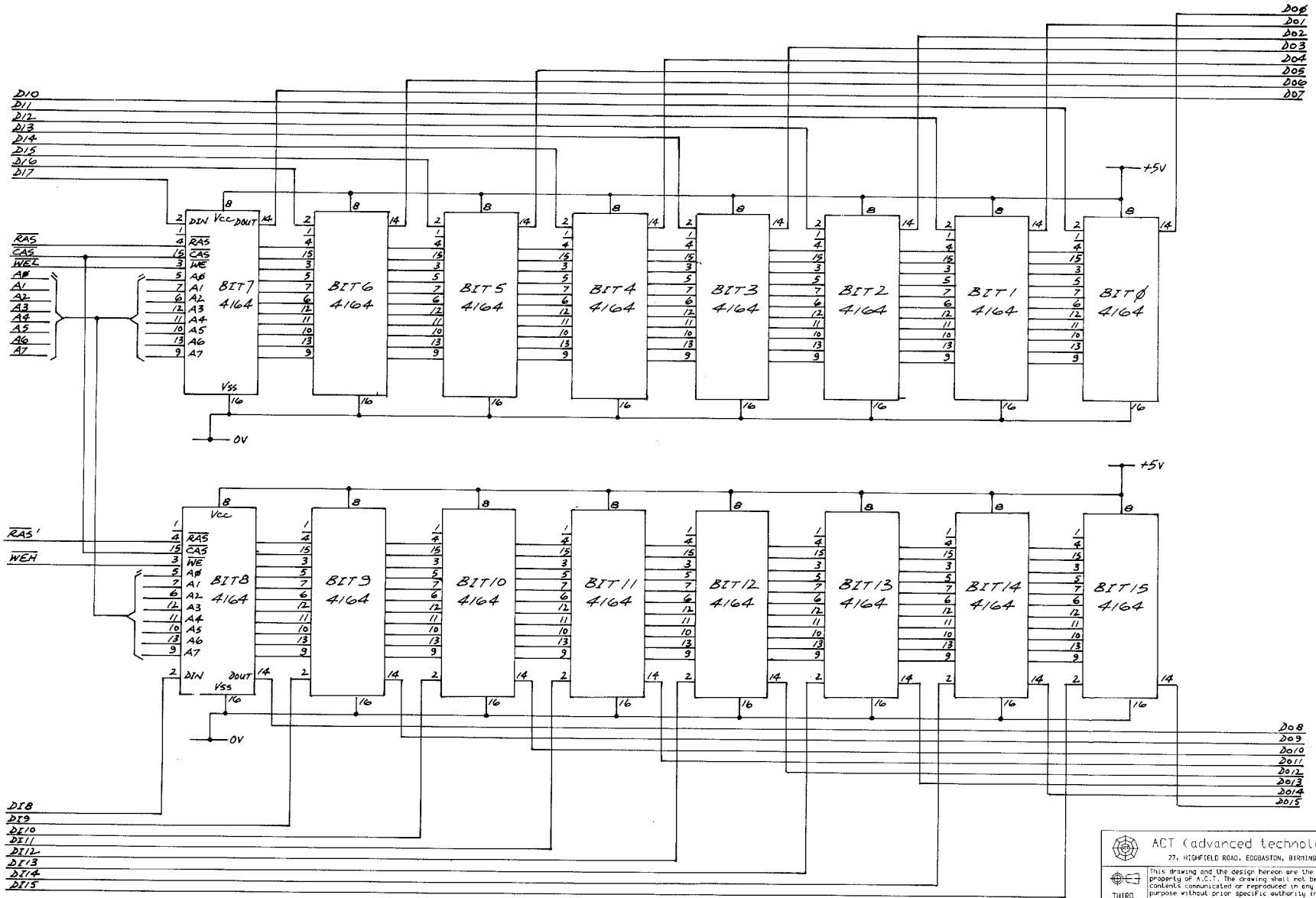
ACT (Advanced Technology) LTD
21, HIGHFIELD ROAD, EGGINTON, BIRMINGHAM
This drawing and the design thereon are the exclusive property of A.C.T. and may not be reproduced nor used for any commercial purposes or communicated in any form whatsoever without prior specific authority in writing of A.C.T.

SCALE
DRAWN
APPROVED
DATE 02-DEC

TITLE: APRICOT F1
CPU AND PERIPHERALS

DRG. NO.

SHT 1 OF 4



NOTE :- 2 BANKS OF 128K RAM SHOWN PER BOARD.

ACT (advanced technology) LTD
27, HIGHFIELD ROAD, EDGBASTON, BIRMINGHAM

This drawing and the design hereon are the exclusive property of A.C.T. The drawing shall not be copied nor its contents communicated or reproduced in any form and for any purpose without prior specific authority in writing of A.C.T.

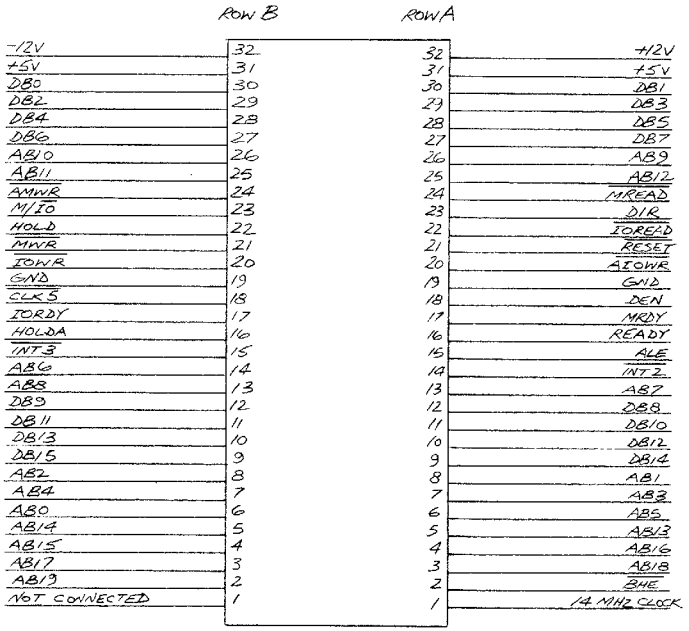
THIRD

SCALE	1:1
DRAWN	
APPROVED	
DATE	02-85

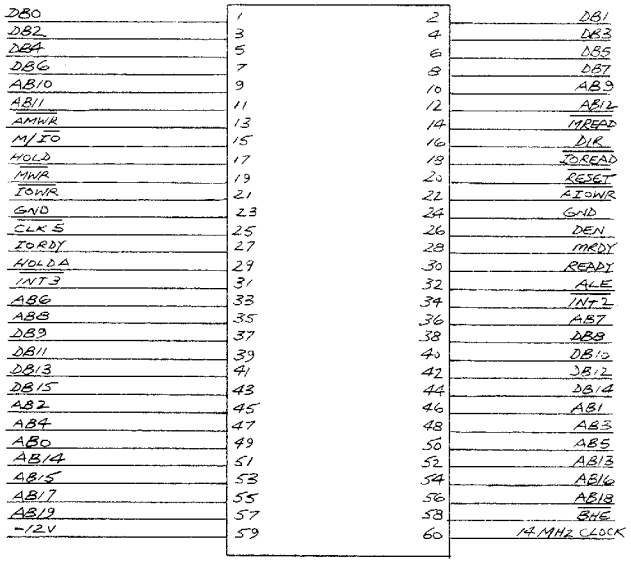
TITLE: APRICOT F1
128K RAM SECTION

DRG. NO.:

SHT 3 OF SHTS 4


ROW B
ROW A
-12V 32 32 +12V
+5V 31 31 +5V
DB0 30 30 DB1
DB2 29 29 DB3
DB4 28 28 DB5
DB6 27 27 DB7
AB10 26 26 AB9
AB11 25 25 AB12
AMWR 24 24 MREAD
M/IO 23 23 DIR
HOLD 22 22 IOREAD
MWR 21 21 RESET
IOWR 20 20 AIOWR
GND 19 19 GND
CLK5 18 18 DEN
IORDY 17 17 MRDY
HOLDA 16 16 READY
INT3 15 15 ALE
AB6 14 14 INT2
AB8 13 13 AB7
DB9 12 12 DB8
DB11 11 11 DB10
DB13 10 10 DB12
DB15 9 9 DB14
AB2 8 8 AB1
AB4 7 7 AB3
AB0 6 6 AB5
AB14 5 5 AB13
AB15 4 4 AB16
AB17 3 3 AB18
AB19 2 2 BHE
NOT CONNECTED 1 1 14 MHz CLOCK

2 x 64 WAY CONNECTORS
(HD7 & 19)



60 WAY CONNECTOR
(HD8)

No.	DESCRIPTION	CHK'D	APP'D	DATE
C	CHANGE NOTE No 069	TD		3/7/84
B	CHANGE NOTE No. 050	TD		11/6/84

REVISIONS

A PRODUCT OF THE UNO RANGE-ENGINEERS CAT. No. 7302

ACT (advanced technology) LTD
27, HIGHFIELD ROAD, EDGBASTON, BIRMINGHAM

THIRD

This drawing and the design hereon are the exclusive property of A.C.T. The drawing shall not be copied nor its contents communicated or reproduced in any form and for any purpose without prior specific authority in writing of A.C.T.

SCALE 1:1
DRAWN
APPROVED
DATE -02-85

TITLE:
APRICOT F1
EXPANSION PORT

DRG. NO.:

SHT 4 OF SHTS 4

594 x 420mm A2

Appendix E ESCape sequence reference table

The ESCape sequence Table in the Screen Driver chapter lists the sequences in Ascii ascending order.

This appendix lists them by the following activities for quick reference:

1. Specials
2. Character attributes
3. Screen attributes
4. Colour
5. Cursor positioning
6. WP primitives
7. Driver environment
8. Keyboard
9. Generic obsoletes

Specials

CHR	HEX	M/C	ACTION
&	26	PFA	Print Page Outputs the contents of the Screen to the line printer. A form-feed is executed first.
'	27	PFA	Print line Outputs the entire line that the cursor is at to a connected line printer. No form feed is executed.
*	2A	A	Change to second character font.
8	38	PFA	Set literal/Test mode ON The escape sequence tells the screen driver to perform the following action on receiving the next character: Ignore the fact that it is a control code (<20H) and print the character associated with it. This means that the font cell characters under (20H) are printed, rather than obeyed. The ESCape must be sent for each character.
?	3F	A	Enter CALC mode This escape sequence switches on the internal BIOS calculator. It is the same as pressing the "calc" key.
[	5B	PFA	ANSI lead-in character. Refer to section ANSI ESCape sequences of the Screen driver chapter for details of the ANSI codes supported.
}	7D	PF	GSX private - this call is documented for completeness only. It is not supported.

Character attributes

CHR	HEX	M/C	ACTION
0	30	PFA	Sets underline mode. All characters printed have a single line of pixels placed under them to simulate underline.
1	31	PFA	Reset underline mode. Cancels the mode set above.
9	39	PFA	Set strikeout mode ON. All following characters are displayed with a horizontal line through them at the centre. This is widely used for deleting data within legal documents.
:	3A	PFA	Set strikeout mode OFF. The code reverts the action of "9" (39H).
p	70	PFA	Enter reverse video mode. All characters printed after this sequence are displayed in inverse video.
q	71	PFA	Cancel reverse video mode. Restores the writing mode to the state it was in before ESC 'p' was executed.

Screen attributes

CHR	HEX	M/C	ACTION
(	28	PFA	Set High intensity Mode Shadow-prints all characters to give the effect of high intensity characters.
)	29	PFA	Set Low intensity Mode Clears the mode set above.
+	2B	A	Clear all high intensity characters.
,	2C	PFA	Set Window size. Takes four parameters in Ascii: <1> Top line + 31 <2> Bottom line + 31 <3> Left-hand column + 31 <4> Right-hand column + 31
-	2D	A	Clear all low-intensity characters.
.	2E	PFA	Reset Window Size. Resets the Window size set by code 2C

Colour

CHR	HEX	M/C	ACTION
5	35	PF	Set foreground colour. This escape sequence takes one parameter which is from "0" (30H) to "?" (3FH). This gives 16 possible indexes. For a list of indexes and colours represented by them see escape sequence "]" (5DH). See also the diagram for ESC "6" (36H) below.
6	36	PF	Set block or background colour. This escape sequence sets the colour of all pixels in the character cell that do not make the actual character shape. As above, it takes one parameter. Diagram of a character cell: ———————— Background. (all 0's) | 00000000 00111100 00100100 001001 ——— Foreground or text 00111100 colour. (all 1's) 00100100 00100100 00000000
]	5D	PF	Set palette code. This escape sequence takes two arguments. The first is the index which needs to be changed, and the second is the colour. e.g. PRINT CHR$(27) "]05" Sets index 0 (in this case the background) to colour 5. Refer to the Colour section of the Screen Driver chapter for full details of index, colour and default Palette settings.

Cursor positioning

CHR	HEX	M/C	ACTION
;	3B	PFA	Position cursor to start of status line. Position the cursor to the start of the status line, i.e. the 25th line of the Screen. The cursor remains on this line until a position cursor command is given.
A	41	PFA	Cursor UP.
B	42	PFA	Cursor DOWN.
C	43	PFA	Cursor RIGHT.
D	44	PFA	Cursor LEFT.
E	45	PFA	Clear screen. The current window is cleared, and the cursor is homed.
H	48	PFA	Home Cursor. The cursor is placed at the top left hand corner of the current text window.
Y	59	PFA	Position Cursor. This sequence takes two parameters. They are the line number and column number in normalised Ascii. e.g. PRINT CHR$(27) "Y" CHR$(10+31) CHR$(15+31) will position the cursor at line 10, column 15.
j	6A	PFA	Save cursor position The current cursor position is noted within the BIOS.
k	6B	PFA	Restore cursor The cursor is restored to the position it was in before ESC 'j' was executed.

WP Primitives

CHR	HEX	M/C	ACTION
@	40	PFA	Enter insert mode. After this escape sequence is issued, whenever a character is printed, all the characters to the right of it will be shifted right one place and the character will be inserted in the space created.
I	49	PFA	Reverse-index and line-feed. This sequence moves the cursor up one line, however if the cursor is at the top of a window then a scroll DOWN of the whole window is performed.
J	4A	PFA	Erase to end of Page. The sequence first of all erases all characters from the cursor position to the end of the current line, and then all subsequent lines below the cursor till the end of the current window or page.
K	4B	PFA	Erase to end of line. This escape sequence erases all characters from the cursor position to the end of the defined right-hand side margin.
L	4C	PFA	Insert line. This sequence places the cursor to the beginning of the current line, and then inserts one line below the current cursor position by scrolling all subsequent lines down by one place.

WP Primitives (Continued)

CHR	HEX	M/C	ACTION
M	4D	PFA	Delete line. This sequence places the cursor to the beginning of the current line and scrolls all lines under it up by one place.
N	4E	PFA	Delete character. The character under the cursor is cleared, and all characters to the right are scrolled left by one position. This is active in the defined right-hand margin space.
O	4F	PFA	Exit Insert Mode. This sequence reverses the effect of ESC "@" (40H)
P	50	PFA	Insert single character. This sequence scrolls all characters from the current cursor position to the defined right-hand margin right by one place.
Q	51	PFA	Scroll left. Takes one parameter which is the number of columns plus 31 that the screen is to be scrolled. This is only active in the current window.
R	52	PFA	Scroll right. As above, except scrolling is right rather than left.
S	53	PFA	Scroll up. As above, but scrolling is UP.
T	54	PFA	Scroll down. As above, but scrolling is down.

WP Primitives (Continued)

CHR	HEX	M/C	ACTION
b	62	PFA	Erase from start of page All characters from the top left-hand corner of the current defined display page size to the current cursor position are cleared.
h	68	PFA	Reverse tab This sequence has the opposite effect of control code 09H. It performs a tabulation operation to the left rather than the right.
l	6C	PFA	Erase line The line which the cursor is on is cleared. Note that no scrolling takes place
o	6F	PFA	Erase to start of current line. All characters from the start of the current line up to and including the character under the cursor are cleared.
v	76	PFA	Wrap at end of line. This escape sequence indicates that the normal screen driver action when the cursor reaches the end of a line should be employed. The action is to return the cursor to the beginning of the next line on the screen.
w	77	PFA	Discard at end of line. When the cursor reaches the end of the current line, it will remain there.

Driver environment

CHR	HEX	M/C	ACTION
7	37	PF	Set screen environment. This sequence is followed by one of the following mode parameters: "0" - Apricot PC & Xi monochrome compatibility "1" - 80 column full colour display "2" - 40 column mode "3"- Apricot PC & Xi monochrome compatibility Refer to section Screen environment in the Screen driver chapter for operational details.
F	46	PFA	Enter VT52 Graphics mode. VT52 mode displays the VT52 graphics character represented by the Ascii value of the Apricot lower case letters. Other characters are not supported.
G	47	PFA	Exit VT52 Graphics mode. Invoke this mode to exit VT52. Failure to do this results in non-lower case letters being incorrectly displayed.
Z	5A	PFA	Identify as VT52. This escape sequence is included as most of the screen driver is DEC VT52 compatible. After issuing this sequence the keyboard buffer is filled with three characters which can be read by an application to determine the device type.

Driver environment (Continued)

CHR	HEX	M/C	ACTION
'	60	PFA	Save environment The first three environment flags are saved. They can then be temporarily changed and restored by another sequence.
a	61	PFA	Restore environment Returns the first three environment flags to their state just after code 60.
x	78	PFA	Set environment flags. Takes one parameter:
		PF	G - reset screen palette
		PF	$ - set to apricot comp mode on default screen
			1 - enable line 25
			2 - nothing
			3 - nothing
			4 - nothing
			5 - cursor off
			6 - nothing
			7 - nothing
			8 - set auto LF on receipt of CR
			9 - set auto CR on receipt of LF
			A - nothing
			B - nothing
			C - nothing
			D - smooth screen scrolling
		P	E - LCD contrast up
		PF	F - bell volume up

Driver environment (Continued)

CHR	HEX	M/C	ACTION
y	79	PFA	Reset environment flags. Takes one parameter:
			1 - disable line 25 2 - nothing 3 - nothing 4 - nothing 5 - cursor on 6 - nothing 7 - nothing 8 - no auto LF on receipt of CR 9 - no auto CR on receipt of LF A - nothing B - nothing C - nothing
		PF	D - fast screen scrolling
		P	E - LCD contrast down
		PF	F - bell volume down
z	7A	PFA	Reset all screen drivers.
			Sets all screen drivers to power-on status.

Keyboard interaction

CHR	HEX	M/C	ACTION
$	24	PFA	Transmit Character Sends the character under the cursor into the keyboard buffer.
4	34	PFA	Change the representation of a key This escape sequence takes 3 paramaters. They are: <1> - Key mode (ascii) : 1 = normal 2 = shift 3 = control <2> - Key number — 1 : 0 = help... etc.(refer to Appendix B) <3> - New key character : Ascii char. or hex equivalent. 10 PRINT CHR$(27)+"4"+"3" +CHR$(72)+"C" This example changes the key with the legend "C" (downcode number 72), in control mode, to generate an ascii "C" (43H) as opposed to a binary 03H. The key has the default attribute of AUTO-REPEAT. Refer to Appendix B for a list of keys, their corresponding character value, down code and attributes. Note: The Screen Driver does not accept non-printable characters (range 0 to 31) in ESCape sequences therefore no Key or Key character can be programmed with a value below 32. See <2> and <3> above.

Keyboard interaction (Continued)

CHR	HEX	M/C	ACTION
/	5C	PFA	Place key in keyboard buffer. This sequence takes one parameter which is placed in the keyboard buffer. If the buffer is full the bell will sound and the character will be ignored. eg: PRINT CHR$(27) "/R" will place an "R" into the keyboard buffer.
n	6E	PFA	Return Cursor position. This sequence places a valid ESC "Y" sequence representing the position of the cursor at the current position in the keyboard buffer. i.e 4 bytes of data: 27+"Y"+(31+column)+(31+row) where column and row have a base of 1.

Generic obsoletes

CHR	HEX	M/C	ACTION
/	2F	A	Set membrane key LED's.
<	3C	A	Display time on MSCREEN.
U	55	A	Enable dual-output to MSCREEN
V	56	A	Disable dual output.
W	57	A	Output text to MSCREEN only.
c	63	A	Disable MSCREEN scrolling.
d	64	A	Enable MSCREEN scrolling.
e	65	A	Switch MSCREEN cursor ON.
f	66	A	Switch MSCREEN cursor OFF.
g	67	A	Disable Time and Data display on MSCREEN.
r	72	A	MSCREEN echo enable.
s	73	A	MSCREEN echo disable.

Appendix F Language interfaces

Overview

Interpretive Basic
The Data Segment Register
Non-BIOS routine calls

Compiled Basic
The Data Segment Register
Non-BIOS routine calls

'C' Programming Language
The Data Segment Register
Non-BIOS routine calls

Overview

The programming examples in this manual are provided in Interpretive Basic.

While it is recognised that the majority of Software products are written in either assembler or compiled High level languages it does not necessarily imply that these tools are universally available.

For this reason, together with the overriding consideration that all examples should be in a common language, Interpretive Basic was chosen.

This appendix is provided to support the programming examples given throughout the manual and as an introduction to interfacing in Microsoft Compiled Basic and 'C'.

Interpretive Basic and the compiled languages mentioned do not provide a compatible means of interfacing to machine code.

The most common problem being that pointers to parameter blocks are either in the incorrect format or on the stack instead of in a register.

Another problem is that, in the case of BIOS, GSX and other installed Device driver calls, pointers need to be set up containing the Applications Data Segment. The address of the Data Segment is not however accessible by many languages.

Apart from these points, languages such as Interpretive Basic have idiosyncrasies which are not at first apparent to the Application programmer with little experience in this field.

Therefore the requirement for interfaces of all kinds and care in using them is needed. This appendix does not attempt to cover all possibilities but merely to support the programming examples given and to act as an introduction to the subject.

The Control device chapter provides details of how to interface to the BIOS in assembler and BASIC. For other external software such as GSX reference to the appropriate manual will provide full programming details.

Interpretive Basic

This is one of the most easily used High Level languages because of it's interactive nature but it has not been developed with a view to interfacing to machine code within the Application bounds.

All of the reasons for this fall outside the scope of this manual however the relevant points are as follows:

1. The use of the Call statement with a string parameter results in the interpreter putting a string descriptor pointer on the stack - not the address of the string. The descriptor is a 3 byte packet containing the string length and the address of the data itself.
2. No functions are available to obtain the value of the Data Segment register.
3. The location of string variables can change during execution of the Application.

As mentioned in the overview many calls to systems software require pointers to parameter blocks.

In addition, the machine code interface must be located within a reserved data area, and this can be achieved by creating it in either a string variable or an integer array. Both may cause problems without due care.

The Data segment is not available because of historical reasons. The only solution is to use an interface or subroutine in machine code to obtain it. The implementation of such an interface is given below.

The location of a string however poses several problems. Strings are by far the most convenient way to store a machine code routine and they are also widely used as parameter blocks.

Interpretive Basic moves strings in memory as it creates them especially when using concatenation.

The following precautions should therefore be taken:

1. Always ensure that the address of a string is correct just prior to calling the machine code routine.
2. Ensure that the string area is not unneccesarily cluttered with work data by executing a FRE(" ") function whenever convenient.

   Microsoft Basic does this from time to time automatically but where concatenations are frequent it is possible that a potential string field overlaps other reserved areas. The machine code routine has no knowledge of the interpreter and may alter the string and corrupt other data.

   In either case caution should be taken to ensure that a string pointer and the contents of the string are unchanged.

The Data Segment register

The routine below creates a machine code interface which when executed returns the value of the Data Segment register in an integer variable. On entry the stack has the following entries:

SP + 4 Address of integer variable
SP + 2 Segment of Return address
SP + 0 Offset of Return address

```
9000 REM routine at GETSEG
9010 DATA &H55,&H56                          :REM push bp:push si
9020 DATA &H89,&HE5                          :REM mov bp,sp
9030 DATA &H8B,&H76,&H08                     :REM mov si,[bp+8]
9040 DATA &H8C,&H1C                          :REM mov [si],DS
9050 DATA &H5E,&H5D                          :REM pop si:pop bp
9060 DATA &HCA,&H02,&H00                     :REM ret 2

9100 REM set up routine
9110 FOR I=1 TO 14
9120   READ CODE%                            :REM get machine code
9130   GETSEG$=GETSEG$+CHR$(CODE%)           :REM store it
9140 NEXT I

9150 GETSEG=PEEK(VARPTR(GETSEG$)+1)+(256*PEEK
     (VARPTR(GETSEG$)+2)

9160 RETURN
```

Notes:

1. To use the above routine do a GOSUB 9100 at the beginning of the program and then call it, for example

   ```
   10 GOSUB 9100
   20 CALL GETSEG(SEG%) ' returns Data Segment in SEG%
   ```

2. Statement 9150 assigns the address of the string GETSEG$ in the variable GETSEG. Strings are usually to be found at the end of the data segments and have offsets greater than 32767. Therefore an integer variable e.g. GETSEG% cannot be used to obtain the address easily.

3. The function VARPTR returns the address of a 3 byte string descriptor of the following format:

   BYTE string length in bytes
   WORD offset to string data (Intel format)

   Statement 9150 picks up the contents of the WORD from the string pointer packet.

Non-BIOS routine calls

Certain calls to installed Device drivers require a Segment:Offset pointer to a parameter block in a register pair.

The routine given below is used to call an external Device driver with a Parameter block in a string variable. The interface extracts the pointer to the parameter block from the string descriptor and places it in the SI register before calling the driver.

On entry the stack pointer is the same as for the example given above.

SP + 4 Address of string descriptor
SP + 2 Segment of Return address
SP + 0 Offset of Return address

Note that the first two Push statements require the offset in the routine to be modified by 4.

```
9200 DATA &H55              : REM push bp
9201 DATA &H56              : REM push si
9202 DATA &H8B,&HEC         : REM mov bp,sp
9203 DATA &H8B,&H6E,&H08 : REM mov bp,[bp+8]
9205 DATA &H8B,&H76,&H01 : REM mov si,1[bp]
9206 DATA &HCD,&Hxx         : REM int xxH
                            : where xx is a valid interrupt address
9206 DATA &H5E              : REM pop si
9207 DATA &H5D              : REM pop bp
9208 DATA &HCA,&H02,&H00 : REM ret 2

9910 CODE$=SPACE$(255)
9920 CODE=PEEK(VARPTR(CODE$)+1)+(256*PEEK
     (VARPTR(CODE$)+2))
9950 FOR I=0 TO 16:READ J%: POKE VI+I,J%: NEXT I: RESTORE
9990 RETURN
```

Notes:

1. The routine is generated and executed by including the following statements in an Application:

```
10 GOSUB 9910                'set up routine
20 CALL CODE(PARAMETER$)     ' calls the routine
```

2. The routine at 9910 should be executed as soon as possible in the Application to ensure that CODE$ is not moved around by the interpreter. If there is doubt then, whilst the Application may be slowed down, recreate the CODE$ each time before using it. As will be seen these problems do not occur in compiled languages.

3. Calls from Interpretive Basic are "Far Calls".

Microsoft Compiled Basic

Microsoft Compiled Basic has the same idiosyncrasies as the interpreted version with the exception that string data is not moved around in the same way.

Therefore the main problems to solve are those of obtaining the Data Segment, String pointers and setting up registers correctly.

The two examples given below are equivalent to those given in the interpretive section.

The Data Segment Register

```
                PAGE 60,132
                TITLE  Get the Data Segment register
code    segment byte    public      'code'
        assume  cs:code
        public  getseg  ;make this routine available to
                         basic
getseg  proc    far     ;must be a "Far" routine
        push    bp      ;save current base page register
        push    si      ;save SI register
        mov     bp,sp   ;copy the stack pointer
        mov     si,8[bp] ;load the address return variable
        mov     [si],ds ;store contents of DS in return
                         variable
        pop     si      ;restore the SI register
        pop     bp      ;restore the base page register
        ret     2       ;return to basic
getseg  endp
code    ends
        end
```

Non-BIOS routines

The task of the interface is to set up the DS:SI registers prior to entering the Device driver. The SI pointer is set from the contents of the string descriptor.

```
        PAGE    60,132
        TITLE   Device driver Interface routine

;
CODE  SEGMENT BYTE      PUBLIC  'CODE'
        ASSUME  CS:CODE
        PUBLIC  IFACE
IFACE  PROC     NEAR
;This routine will be called with the following
;stack contents.

;               STACK: param. block offset : +2
;              (SP) ->: return address     :  0

        PUSH    BP                  ;save working
                                     registers
        PUSH    SI
        MOV     BP,SP               ;point to stack.
                                     (param.block offset
                                    ;will now be 6 bytes
                                     down the stack)
        MOV     BP,6[BP]            ;BP points to string
                                     descriptor
        MOV     SI,1[BP]            ;DS:SI now point to
                                     paramater string
        INT     0xxh                ;call the device at
                                     interrupt xxH
        POP     SI                  ;restore the working
                                     registers
        POP     BP
        RET     2

IFACE  ENDP
CODE   ENDS
        END
```

Microsoft 'C' Programming Language

Unlike interpretive and Compiled Basic, the 'C' programming language provides pointers to Parameter blocks in the correct form and also offers a choice of Near or Far calls.

Interfaces are still required as for other languages to obtain the Data Segment to support calls to the BIOS.

Examples of the interfaces are given below. They should be assembled and linked to the Application.

The Data Segment Register

```
        PAGE 60,132
        TITLE   Get the Data Segment register

pgroup  group   code
code    segment byte        public      'code'
        assume  cs:pgroup
        public  getseg      ;make this routine available
                             to basic
getseg  proc    far         ;routine may be "Far or Near"
        push    bp          ;save current base page register
        push    si          ;save SI register
        mov     bp,sp       ;copy the stack pointer
        mov     si,8[bp]    ;load the address return
                             variable
        mov     [si],ds     ;store contents of DS in
                             return variable
        pop     si          ;restore the SI register
        pop     bp          ;restore the base page register
        ret     2           ;
getseg  endp
code    ends
end
```

```
        PAGE    60,132
        TITLE   Device Driver Interface Program
;
PGROUP  GROUP   PROG
PROG    SEGMENT BYTE    PUBLIC  'PROG'
        ASSUME  CS:PGROUP

        PUBLIC  IFACE
IFACE   PROC    NEAR
;This routine will be called with the following
;stack contents:
;               STACK: param. block offset : +2
;               (SP) -> : return address    :  0
        PUSH    BP              ;save working
                                 registers
        PUSH    SI
        MOV     BP,SP           ;point to stack.
                                 (param.block offset
                                ;will now be 6 bytes
                                 down the stack)
        MOV     SI,6[BP]        ;DS:SI now point to
                                 paramater block
        INT     0xxh            ;call the device driver
                                 with interrupt xxH
        POP     SI              ;restore the working
                                 registers
        POP     BP
        RET     2
IFACE   ENDP
PROG    ENDS
        END
```

Index

Index

A

ANSI escape sequences *3.3/4, 3.3/13*
Application
  interrupts *3.1/11*
  RAM *2.2/7*
  interface *1.2/4*
Apricot compatibility *3.3/6, 3.4/21*
ASCII
  control codes *3.3/18*
  screen images *3.1/9, 3.1/20*
Assembler *3.2/3*
ASSIGN.SYS *3.10/18*
Asynchronous
  communications *2.2/19, 2.7/41*
  mode *2.7/2, 2.7/6*
Audio
  output *2.7/2, 2.10/4*
  parameters *2.10/7*
  tones *2.10/5*
Auto repeat *3.4/4*
Auxiliary audio input *2.10/4*

B

Bandpass filter *2.10/4*
BASIC *3.2/2*
  escape sequences *3.3/5*
  interface Appx f
Batteries *2.12/3*
Baud rate
  reception *3.5/8*
  transmission *3.5/8*
Bell *2.10/2, 3.3/18, 3.4/4, 3.4/19, 3.8/1*
BIOS *1.2/2, 2.7/33, 2.7/35, 2.10/2, 3.1/2, 3.9/2*
  code *2.2/8*
  configuration table *3.1/9*
  data *2.2/7*
  default constants *3.1/9*
  initialisation *3.1/7*
  parameter block *3.9/4*

pointers *3.1/9, 3.1/17*
details *3.1/18*
format *3.1/17*
table *3.1/18*
stack *2.2/7, 3.1/9*
working area *2.2/7*
Bisync *2.7/3, 2.7/6, 2.7/25*
Bit
mapped RAM *2.2/6*
screen image *3.1/20, 3.3/4*
synchronous comms. *2.2/19*
synchronous mode *2.7/3*
Boot
disk *3.1/3*
label sector *3.1/4, 3.4/4, 3.5/3, 3.9/2, 3.9/4*
ROMS *2.2/8*
routines *3.3/6*
Bootstrap loader *2.2/8, 3.1/3*
BPB *3.9/4*
Byte synchronous
communications *2.2/19*
mode *2.7/3*

C

C language interface *Appx F*
Calculator software *2.2/8, 3.1/8, 3.2/3*
CAPS LOCK *3.4/4*
Centronics interface *2.8/2, 2.2/20, 2.8/2*
Character
attributes *2.4/5, 3.3/4, 3.3/34*
font *1.2/13, 3.1/7, 3.1/9*
font default Appx C
Circuit diagrams Appx D
Clock driver *1.2/16, 3.1/5, 3.1/7, 3.2/23, 3.7/1*
Cluster *3.9/4*
Cold start *3.1/3*
Colour
ASCII index *3.3/10*
ASCII code *3.3/10*
palette *3.3/8*
Communications *1.1/18*
Compiled BASIC *3.2/3*
interface Appx F
Composite video *2.2/13, 2.4/7*
signal *2.4/6*
socket *2.4/30*

CONFIG.SYS *1.2/2*
Configuration table *3.1/21, 3.3/10, 3.3/36, 3.4/22, 3.5/3*
Configurator *3.4/20*
Connectors *2.1/4*
Control device *1.2/4, 1.2/10, 3.1/2, 3.1/7, 3.2/2*
  access *3.2/4, 3.2/6*
  errors *3.2/7*
  parameters *3.2/3*
Control key *2.12/4, 2.12/10, 3.4/12*
Controller programming *2.3/7, 2.3/13*
Cooling *2.1/4*
Corvus Omninet *1.3/8*
CTC (Counter/Timer) *2.7/2, 2.7/38, 2.9/2, 2.10/2*
  address allocation *2.9/8*
  channel 0
    exp. bus interrupt *2.9/10*
  channel 1
    RS232 baud rate *2.9/11*
  channel 2
    sound frequency *2.9/14*
  channel 3
    system clock interrupt *2.9/16*
  channel modes *2.9/4*
  channel usage *2.9/7*
  clock rates *2.9/5*
  components *2.9/4*
  downcounter *2.9/4*
  functions *2.9/2*
  initialisation *2.9/9*
  interrupt *2.9/5*
    vector *2.9/10*
  programming *2.9/9*
  return from interrupt *2.9/16*
  time constant *2.9/4*
  zero count *2.9/4*
Cursor control *3.3/4*
  movement *3.3/18*
Cyclic redundancy check
2.6/6, 2.7/6, 2.7/20, 2.7/23, 2.7/29

D

Date key *2.12/10, 2.12/13*
Date/time clock *3.7/1*
DC distribution *2.1/11*
Device driver *1.2/2, 1.2/11*
  setting *3.2/8*
  numbers *3.2/9*
Diagnostics *2.2/8, 3.1/3*
Digital Research *3.10/2*
Disk
  70 track *2.11/2, 3.9/4*
  80 track *2.11/13*
  cluster *3.9/4*
  control *2.2/15*
  controller *2.1/7, 2.2/3, 2.6/3*
  drive *1.1/9, 2.1/3, 2.1/9*
    combinations *3.9/4*
    configuration *3.9/9*
    connections *2.6/37*
    control *2.6/14*
    controller connections *2.11/2*
    mechanism *2.11/9*
    motor control *2.6/18, 2.11/11*
    packing disks *2.11/2*
    second *2.1/5, 2.11/2*
    select control *2.11/10*
    selection *2.6/18*
    specification *2.11/12*
    switch setting *2.11/10*
    types *3.9/4*
  driver *1.2/15, 3.1/5, 3.2/26, 3.9/2*
  eject button *2.1/9*
  electronics *2.11/4*
  format *2.6/2, 2.11/14, 3.9/7*
  format for MS-DOS *3.9/6*
  formatting *2.6/7*
  formatting commands *2.6/28*
  head loading *2.6/19, 2.11/9*
  head positioning *2.6/8, 2.6/19, 2.11/9*
  head position commands *2.6/8*
  head select *3.7/2*
  indicator *2.1/9*
  insertion/removal *2.11/14*
  interface connection *2.11/6*
  interface details *2.6/2, 2.11/4*
  label sector details *3.1/22*

MFM encoded data *2.11/4*
MS-DOS format *3.9/2*
non-ACT *2.11/13*
physical format *3.9/2*
precautions *2.11/13*
read *2.6/6*
read address *2.6/29*
read track *2.6/29*
read/write heads *2.11/9*
restore command *2.6/20*
sensors and detector *2.11/10*
swapping *3.9/2, 3.9/8*
track format *2.6/28, 2.6/39*
track image *2.6/31, 2.6/31*
transportation *2.11/2*
write *2.6/6*
write protect *2.11/14*
write track command *2.6/30*
Display
access sequence *2.4/28*
coding 2 colours *2.4/19*
coding 4 colours *2.4/21*
coding 8 colours *2.4/19*
coding monochrome *2.4/19, 2.4/24*
connectors *2.4/29*
control *2.2/11, 2.2/12*
circuitry *2.4/7*
schematic *2.4/6*
drive signals *2.4/6*
features *1.1/10*
memory *2.2/11, 2.4/3*
memory planes *2.4/9*
mode selection *2.4/24*
modes *1.2/13, 2.4/3*
plane coding *2.4/18*
RAM *2.4/6, 2.4/7, 2.4/9*
RAM access control *2.4/27*
refresh control *2.4/25*
scrolling *2.4/5, 2.4/6*
timing ROM *2.4/8, 2.4/27*
timing signals *2.4/8*
word *2.4/10*
decoding 80 column *2.4/22*
DMA facilities *2.2/17*
Double sided disks *2.2/15, 2.6/2*

Downcode *1.2/11, 3.4/2*
handler *3.4/17*
table *3.4/23*
Dummy interrupt handler *3.1/7*

E

Escape sequence *1.2/14, 3.3/4*
ANSI *3.3/13*
for palette setting *3.3/11*
in BASIC *3.3/5*
strings *3.3/5*
table *3.3/4, 3.3/18, Appx E*
Expansion
board *1.1/20, 1.3/5, 2.2/16*
layout *2.5/16*
bus *2.2/6*
connector *2.5/7*
slot *2.2/13, 2.5/4, 2.5/12*
addresses *2.5/12*
compatible pins *2.5/4*
electrical spec. *2.5/8*
I/O space *2.5/12*
interrupt *2.5/13*
interrupt set-up *2.5/14*
pin detail *2.5/9*
unit *1.3/11, 2.1/9, 2.2/16, 2.2/17, 2.5/7*

F

FDC (Floppy disk cntrl.)
command register *2.6/12, 2.6/15*
commands *2.6/15*
data register *2.6/16*
data requests *2.6/11*
force interrupt command *2.6/33*
interrupt flags *2.6/34*
interrupt requests *2.6/12*
pin definition *2.6/4*
programming *2.6/17*
read sector command *2.6/25*
registers *2.6/11*
sector register *2.6/16*
seek command *2.6/21*
status register
2.6/12, 2.6/15, 2.6/16, 2.6/22, 2.6/26, 2.6/32

step command *2.6/21*
step-in command *2.6/21*
step-out command *2.6/21*
system connections *2.6/35*
track register *2.6/16*
write sector command *2.6/24*
Fibre optic link *2.12/3*
File data transfers *2.6/23*
Fixed position files *3.1/20*
Fonts *2.4/4*
alteration *3.3/16*
edit utility *1.2/14, 3.3/16*
pointer *2.4/4, 3.3/16*
table *3.3/6*
Frame period *2.4/13*
Front panel *2.1/6*
Fuse *2.1/11*
rating *2.1/12*

G

GDOS *3.10/2*
calls *3.10/5*
Generic
BIOS *3.5/3*
interface *3.1/2*
keyboard *3.4/2*
GIOS *3.10/2*
driver default *3.10/18*
Graphics
driver *1.2/3*
extension system *3.10/2*
interface *1.2/2, 1.2/6*
GRAPHICS.EXE *3.10/18*
GSX *3.10/2*
1.3 additions *3.10/6*
86 *3.10/2*
bit block move *3.10/8*
cell arrays *3.10/5*
colour *3.10/4*
cursor *3.10/14*
escapes *3.10/5*

fill
attributes *3.10/4*
pattern *3.10/10*
perimeter *3.10/10*
rectangle *3.10/12*
style *3.10/9*
general drawing *3.10/4*
input
choice *3.10/5*
devices *3.10/5*
locator *3.10/5*
string *3.10/5*
valuator *3.10/5*
line attributes *3.10/3*
line style *3.10/7*
marker attributes *3.10/3*
mouse button *3.10/16*
mouse form *3.10/13*
Prestel operations *3.10/16*
system files *3.10/18*
text attributes *3.10/3*

H

Hamming code *3.4/2*
Hamming format *2.12/7, 2.12/10*
Handling key codes *1.2/12*
Hard copy *3.3/4*
Hardware
access *1.1/5*
drivers *3.1/7, 3.2/2*
interrupts *3.1/16*
Help key *3.4/16*

I

Infra-red
detector board *2.1/8, 2.2/19*
detectors *2.1/3*
pulse conversion *2.1/8*
signal *2.1/8*
transmission *2.1/3*
Initialised drivers *3.1/5*
Internal configuration *3.1/4*
Internal expansion slot *2.1/9*

Interrupt
control *2.2/10, 2.3/9*
details *3.1/12*
disable *2.2/10, 2.3/3*
enable *2.2/10, 2.3/3*
pointer location *3.1/11*
priority *2.2/10, 2.3/2, 2.3/5*
requests *2.2/10*
status reset *2.3/12*
vector format *3.1/11*
vectors *2.2/7, 2.3/3, 2.3/6,,2.3/11, 3.1/7, 3.1/9,*
IRGB output *2.4/6*

K

Key
click *3.8/1*
code prefixes *3.4/14*
decoding *3.4/2*
edit utility *3.4/6*
prefix *3.4/4*
ring buffer *3.4/4*
string pointer *3.4/3, 3.4/3*
strings *3.4/8*
strings default *3.4/13*
switch array *2.12/3*
tops *2.12/3*
Keyboard *1.1/13, 2.12/2, 3.4/2*
attributes *3.4/3*
batteries *2.12/3*
circuitry *2.12/4*
click *2.10/2*
data *2.2/17, 2.7/2*
encoding *2.12/6*
format *2.7/32, 2.12/5*
steering *3.4/16*
driver *1.2/11, 3.1/5, 3.2/12, 3.3/6, 3.4/2,*
alteration *3.4/10*
initialisation *3.4/15*
hamming format *2.2/18, 2.12/7*
I.R. LEDS *2.12/3*
lock *3.4/16*
mechanics *2.12/3*
multi-key closure *2.12/5*
power supply *2.12/2*
queue *3.4/19*
scanning *2.12/4*

serial packet *2.2/18*
sleep mode *2.12/5*
strings *1.2/12*
synchronous format *2.12/7*
table *3.1/7, 3.1/9, 3.4/2*
table alteration *3.4/6*
table default *1.2/11, Appx B*
table save *3.4/7*
transmission format *2.12/6*
user interrupt *3.4/14*
x/y co-ordinates *2.12/9*
Key edit utility *1.2/12*

L

Language interfaces *Appx F*
Light pipe *2.1/6, 2.1/8, 2.12/2*
Local key *3.4/4*
Logic circuitry *2.1/3*
Loudspeaker *2.1/3, 2.2/21, 2.10/4*

M

Machine code subroutines *3.2/5*
Mains filter *2.1/10*
Mains frequency *1.2/13*
Maskable interrupts *2.3/10*
Memory *1.1/8, 2.2/7*
Memory map *2.2/9, 3.1/9*
Modem board *1.3/6, 2.7/6, 2.7/24, 2.7/26, 2.7/38*
Modem driver *1.2/3, 2.2/16*
Monitor transformer *2.1/10*
Monochrome monitor *2.1/10*
Monosync *2.7/2, 2.7/6, 2.7/25, 2.10/4, 2.12/7*
Mouse
data *2.2/17, 2.7/2, 3.4/2, 3.4/16, 3.5/2*
data format *2.7/32*
driver *1.2/3, 1.3/9, 3.2/22*
serial packet *2.2/18*
MS-DOS *1.2/2, 1.2/4, 1.3/8, 1.3/10, 3.1/9, 3.2/2, 3.4/2*
disk format *3.9/6*
MS-NET *1.3/8*
Multi-key closure *2.12/5*

N

NMI *2.2/3, 2.2/11, 2.2/16, 2.2/17, 2.3/4*
Non MS-DOS systems *3.9/3*

O

Op. system interface *1.2/8*

P

PABX network *1.3/7*
Palette
  RAM *2.2/8, 2.4/8, 2.4/10, 2.4/15, 3.3/8*
  programming *2.4/18*
  setting sequences *3.3/11*
  Parallel I/O
  configuration *3.6/2*
  driver *1.2/15, 3.2/20, 3.6/2*
  status lines *3.6/2*
Parallel printer port *2.2/20*
Pascal *3.2/3*
Photodiode *2.1/8*
Physical dimensions *2.1/12*
Pixel image *2.2/11, 2.4/8, 3.1/20, 3.3/34,*
Point 32 LAN *1.3/8*
Port addresses *2.2/22*
Power supply unit *2.1/3, 2.1/10*
Pre-boot arrow *3.7/2*
Printer
  addresses *2.8/6*
  connector *2.8/5*
  data port *2.8/7*
  data strobe *2.8/7*
  data transfers *2.8/4*
  driver *3.1/5*
  interface *2.8/3*
  status *2.8/7*
  support *1.1/1*
Processing capability *1.1/6*
Processor, 8086 *2.1/7, 2.2/3, 2.2/6, 2.2/15*

R

RAM
  banks *2.2/7*
  BIOS *3.1/2, 3.1/20, 3.2/2, 3.4/4*
  cards *2.2/16*
  disk *1.2/2*
  expansion *2.2/7*
Register copy table *2.7/35*
Repeat rate key *2.12/10, 2.12/12*
Reset key *2.2/18, 2.12/10, 2.12/11*
Return from
  interrupt sequence *2.2/10, 2.3/3, 2.3/12*
ROM
  BIOS *1.2/2, 1.2/4, 1.2/10, 1.3/10, 3.1/2, 3.1/4*
    data *3.1/9*
RS232
  baud rate *2.9/13, 3.5/2*
  communications *2.2/19, 2.7/2, 2.7/38*
  connector *2.7/39*
  driver *1.2/15, 3.1/5*

S

Scan line modes *2.4/4*
Screen
  bit image *2.2/7, 3.3/33*
  bit image map *3.3/34*
  bit plane *3.3/33*
  blanking *2.4/6*
  character attributes *3.3/4*
  character word *3.3/7*
  cursor addressing *3.3/15*
  driver *1.2/12, 3.1/5, 3.2/10, 3.3/2*
  dump *3.3/4*
  environment *3.3/10*
  images *3.3/3*
  modes *3.3/6, 3.3/7, 3.3/35*
  scrolling *3.3/4, 3.3/35*
  windows *3.3/15*
Scrolling effects *2.2/15*
Serial
  I/O driver *3.2/14, 3.5/2*
  driver differences *3.5/6*
  printer configuration *3.5/4*
SET TIME key *2.12/10, 2.12/12, 3.1/8, 3.4/16*
SHIFT key *2.12/4, 2.12/10, 3.4/12*

SHIFT LOCK *3.4/4*
Simple noise *2.10/5*
Single-sided disks *2.1/9, 2.2/15, 2.6/2*
SIO (Serial I/O)
2.7/2, 2.8/3, 2.8/7, 2.10/2, 2.10/4, 3.4/2, 3.5/2
address search *2.7/20*
architecture *2.7/7*
auto enable *2.7/21*
base vector *2.7/41*
block diagram *2.7/4*
clock rate *2.7/22*
configuration table *3.5/8*
copy registers *2.7/35, 2.7/40*
driver *3.5/2*
configuration *3.5/3*
differences *3.5/6*
end of frame *2.7/29*
error reset *2.7/8*
external interupt *2.7/17*
FIFO stack *2.7/8*
hunt mode *2.7/21*
interrupt *2.7/7, 2.7/9, 2.7/26,2.7/29*
priority *2.7/30, 2.8/7, 3.4/2*
sequence *2.7/30*
vector *2.7/18*
mode *2.7/22*
overview *2.7/5*
parity *2.7/22*
pin detail *2.7/45*
port addresses *2.7/11*
port A
connections *2.7/47*
programming *2.7/35*
write register 1 *2.7/37*
write register 3 *2.7/37*
write register 4 *2.7/36*
write register 7 *2.7/37*
port B
connections *2.7/49*
interrupt *2.7/19*
programming *2.7/40*
write register 1 *2.7/44*
write register 2 *2.7/41*
write register 3 *2.7/44*
write register 4 *2.7/42*
write register 5 *2.7/43, 2.7/43*
processor interface *2.7/11*

read registers *2.7/8, 2.7/25*
read register 0 *2.7/26*
read register 2 *2.7/29*
ready controls *2.7/19*
receive
bits *2.7/21*
characters *2.7/26*
enable *2.7/20*
interrupt *2.7/19*
path *2.7/8*
RTS *2.7/23*
send break *2.7/24*
status affects vector *2.7/17, 2.7/18*
sync
bits *2.7/25*
character *2.7/20*
mode *2.7/22*
system connections *2.7/45*
transmit
bits *2.7/24*
buffer *2.7/26*
enable *2.7/24*
interrupt enable *2.7/17, 2.7/18*
path *2.7/10*
user interrupts *3.5/6*
write register *2.7/7, 2.7/12*
0 *2.7/13*
1 *2.7/17*
2 *2.7/19*
3 *2.7/20*
4 *2.7/21*
5 *2.7/23*
6 *2.7/25*
7 *2.7/25*
aux resets *2.7/16*
commands *2.7/13, 2.7/15*
initialise *2.7/36*
pointers *2.7/13*
summary *2.7/13*
update *2.7/35*
Soft-sectoring *2.6/2*
Software interrupt *1.2/6, 3.1/11*

Sound
address allocation *2.10/6*
complex
duration *2.6/10*
frequency *2.6/10*
waveform *2.10/5, 2.6/10*
driver *3.2/24, 3.8/1*
timeout *3.7/2*
duration *2.10/9, 2.6/10*
frequency *2.10/9*
generation *2.2/21, 2.7/34, 2.7/37, 2.10/2, 2.10/5*
programming *2.10/7*
initialisation *2.10/8,*
simple tones *2.10/8,*
volume *2.10/9,*
waveform shape *2.10/9,*
Special keys *1.2/11, 2.12/10, 2.12/11, 3.4/12*
Specification *1.1/21*
Standard features *1.1/2*
Start up
screen *3.1/3*
sequence *2.2/18*
Stepping motor rate bits *2.6/20*
Synchronous mode *2.7/2, 2.7/6*
Synthesized sounds *2.7/2, 2.10/5*
Sysinit *3.1/9*
System
board *2.1/3, 2.1/6, 2.2/3*
data schematic *2.2/4*
clock *2.2/10, 2.2/17*
interrupt *2.9/2*
mechanics *2.1/3*
memory *2.5/12*
RAM *2.2/7, 2.4/3*
RAM map *2.2/7*
reset *2.2/18*
timer *2.2/21*
Systems unit schematic *2.1/12*

T

Telephone network *1.3/6*
Text
attributes *3.3/8*
with graphics *2.2/11, 2.4/3*
TIME DATE key *2.12/10, 2.12/13, 3.1/3, 3.1/8, 3.4/16*
TIME — SET key *2.12/10, 2.12/12*

Track
format *2.6/2*
position *2.6/8*
register *2.6/8*
TV modulator *2.2/13, 2.4/6, 2.4/7, 2.4/30*

U

User
interrupt *3.4/14, 3.5/2, 3.5/6, 3.7/1*
RAM *2.2/7, 3.3/6*

V

Video
control signals *2.4/25*
line pointers *2.2/7, 2.2/11, 2.4/3, 2.4/5, 3.1/20, 3.3/33*
pointer RAM data *2.4/12, 2.4/15*
signals *2.2/12*
Virtual screen *1.2/14*

W

Wait states *2.2/6*
Warm start *3.1/3*
Winchester
controller *2.2/16, 3.1/3, 3.9/2, 3.9/4*
driver *1.2/17, 1.3/10, 3.1/6, 3.2/28*
formatting *3.9/7*
head select *3.7/2*
Word processing
primitives *3.3/4*

Z

Z80
CTC *2.1/7, 2.2/3, 2.2/10, 2.2/19, 2.2/21, 2.3/2*
registers *2.3/8*
SIO *2.1/7, 2.1/8, 2.2/3, 2.2/10, 2.2/17, 2.2/19, 2.2/21, 2.3/2*
registers *2.3/8*

Addenda